UNIVERSITY OF TWENTE.

Matching Profiles from Social Network Sites

Similarity Calculations with Social Network Support

Master Thesis of Irma Veldman
Computer Science,
Track Information Systems Engineering
Enschede, October 23, 2009

Supervisors University of Twente
Ander de Keijzer
Maurice van Keulen

Supervisors Topicus FinCare
Jasper Laagland
Wouter de Jong

Matching Profiles from Social Network Sites

Similarity Calculations with Social Network Support

Irma Veldman

October 23, 2009

Samenvatting

De laatste jaren zijn sociale netwerk sites enorm populair geworden. Veel mensen zijn lid van één of meer van deze profielen sites en zetten veel persoonlijke informatie online. Deze, vaak publiekelijk beschikbare, informatie kan nuttig zijn voor veel verschillende doeleinden. Het bij elkaar voegen van alle informatie van een persoon tot één profiel maakt dit nog waardevoller. Ontdekken welke profielen bij dezelfde persoon horen is daarmee heel belangrijk geworden. Dit probleem wordt Entity Resolution (ER) genoemd.

In dit onderzoek ontwikkelen we een model dat het ER-probleem in het geval van profielen van sociale netwerk sites zal oplossen. Eerst presenteren we een simpel model. Daarna proberen we dit model te verbeteren door gebruik te maken van de sociale netwerken die een gebruiker kan hebben op deze sites. Wij zijn van mening dat het model significant verbeterd kan worden door deze netwerken in het model op te nemen.

Het algemene idee is als volgt: we hebben twee sites met profielen. Door toepassing van het model proberen we uit te vinden welke profielen van de ene site overeenkomen met welke profielen van de andere site. Aanname hierbij is dat een persoon hoogstens één profiel per site heeft.

In het simpele model vergelijken we eerst alle profielen van de ene site met alle profielen van de andere site. Voor ieder paar van profielen levert dit een score op: de paarsgewijze gelijkheidsscore. Hoe hoger deze score, hoe groter de kans dat deze profielen tot dezelfde persoon behoren. De paren die voldoen aan de paarsgewijze drempel zijn kandidaat-match. Uit alle kandidaat-matches worden uiteindelijk de matches gekozen.

Het netwerkmodel begint op dezelfde manier. Nadat de kandidaat-matches zijn vastgesteld, begint de netwerkfase. Voor elke kandidaat-match wordt dan de netwerkgelijkheidsscore bepaald. Deze score wordt berekend door het bepalen van de overlap die er is tussen de netwerken van de beide profielen in de kandidaat-match. Hoe groter de overlap, hoe hoger de netwerkgelijkheidsscore en hoe groter de kans dat deze profielen tot dezelfde persoon behoren. Deze keer moeten de kandidaat-matches ook nog voldoen aan de netwerkdrempel om kandidaat-match te blijven. Uit alle overgebleven kandidaat-matches worden dan de matches gekozen.

Om te testen of het netwerkmodel inderdaad betere resultaten oplevert dan het simpele model, hebben we experimenten opgezet. Omdat er geen goede datasets beschikbaar waren, hebben we zelf een dataset verzameld. Helaas had deze dataset wel wat beperkingen. Daarnaast hebben we een prototype gebouwd dat het model implementeert. Het prototype bevat verschillende parameters waarvan de waarden tijdens de experimenten gevarieerd kunnen worden om een

goede configuratie te vinden.

Het netwerkmodel legt meer voorwaarden op voor een paar om als match aangemerkt te kunnen worden. De experimentele resultaten bevestigen dit. Dat betekent dat de precisie van de resultaten omhoog gaat. Aan de andere kant worden er, door deze stricte voorwaarden, ook overeenkomende profielen gemist wat zeer ongewenst is. Echter, in het geval dat er profielen in de dataset zitten die ambigu zijn, kan het netwerkmodel onderscheiden welke profielen een correcte match zijn en welke niet. Deze situatie zal vaak voorkomen in de echte wereld en daarom denken wij dat het netwerkmodel een goede bijdrage kan leveren aan het oplossen van het ER-probleem, in dit specifieke geval.

Summary

In recent years social networking sites have become very popular. Many people are member of one or more of these profile sites and tend to put a lot of personal information online. This often publicly available data can be useful for many purposes. Retrieving all available data from one person and merging it into one profile even more. Detection of which profiles belong to the same person becomes very important. This task is called Entity Resolution (ER).

In this research we develop a model to solve the ER problem for profiles from social networking sites. First, we present a simple model. Then, we try to improve this model by making use of the social networks a member can have on these sites. We believe that involving the networks can improve the results significantly.

General idea is that we have two sites with profiles. With the model we try to find out which profiles of the first profile site correspond to which profiles of the second profile site, whereby we assume a person to have at most one profile at each profile site.

In the simple model, we compare all profiles of the first profile site against all profiles of the second site. This comparison will result in a score for each pair: the pairwise similarity score. The higher this score, the higher the probability that these profiles belong to the same person. The pairs that satisfy the so-called pairwise threshold are the candidate matches. From these candidate matches, the matches are chosen.

In the network model, we start the same way. When the list of candidate matches is determined, the network phase is started. For each candidate match the network similarity score is calculated. This is done by determining the overlap in the networks of both profiles in the candidate match. The more overlap between the networks, the higher the network similarity score, the higher the probability that the profiles in the candidate match belong to the same person. This time, the candidate matches should satisfy a network threshold in order to stay a candidate match. Then, from the remaining candidate matches the matches are chosen.

In order to test whether the network model would indeed improve the simple model, we have set up experiments. Since no suitable data sets were available, we retrieved our own data set. Unfortunately, it appeared to have some limitations. Also, we have built a prototype that implemented the model. The prototype has several parameters for which we could vary the values in the experiments to find a good configuration.

The network model ensures that there are more conditions that need to be met to be a match. The experimental results confirm this. That means that

the precision of the results increases. On the other side, due to these strict conditions, corresponding profiles are missed, which is undesired. However, in case there are ambiguous profiles in the set, the network model can distinguish the correct profile. This situation will occur frequently in real life, hence we think the network model can really contribute to solving the ER problem.

Preface

This master thesis is the result of the graduation of the study Computer Science at the University of Twente. The assignment was conducted at Topicus, a young and innovative ICT company in Deventer.

I would like to thank everyone who has provided input and ideas. In particular, I would like to thank my supervisors at Topicus: Jasper for his daily guidance and his progressive views on the topic, and Wouter for his guidance on the direction and structure. I would like to thank Ander and Maurice, my supervisors at the university, for their constructive criticism and input. Working with all four was very enjoyable.

Besides my supervisors, I would like to thank my colleagues for the pleasant working atmosphere.

Finally, I would like to thank my family, my parents and Elmer in particular, for their support during the past half year and during my entire study.

Irma Veldman
Enschede, October 2009

Contents

Samenvatting v

Summary vii

Preface ix

1 Introduction 1
- 1.1 Research . . . 2
  - 1.1.1 Research Goal and Scope . . . 2
  - 1.1.2 Research Questions . . . 3
  - 1.1.3 Research Approach . . . 3
  - 1.1.4 Contributions . . . 4
- 1.2 Terminology . . . 4
- 1.3 Organization . . . 4

2 Data Sources 7
- 2.1 Profile Sites . . . 7
- 2.2 Issues with Generic Data Retrieval . . . 11
  - 2.2.1 Schema Mappings . . . 11
- 2.3 Retrieved Data Set . . . 13
  - 2.3.1 Availability of Data . . . 13
  - 2.3.2 Data Format . . . 14
  - 2.3.3 Crawling the Internet . . . 16
  - 2.3.4 Ground Truth . . . 17
  - 2.3.5 Uncertainty of Data on Profile Pages . . . 17

3 Simple Profile Matching 19
- 3.1 Syntactical Similarity . . . 20
  - 3.1.1 Approximate String Matching . . . 21
    - 3.1.1.1 Character-based Methods . . . 21
    - 3.1.1.2 Token-based methods . . . 21
    - 3.1.1.3 Hybrid methods . . . 22
  - 3.1.2 Examples . . . 22
- 3.2 Semantical Similarity . . . 23
  - 3.2.1 Ontologies . . . 23
- 3.3 Profile Matching Model . . . 24
- 3.4 Prototype . . . 29
  - 3.4.1 Pairwise Comparison . . . 29

3.4.2 Determining Matches . . . 30
3.5 Concluding Remarks . . . 30
3.5.1 Expectations . . . 31

4 Profile Matching with Social Network Support 33
4.1 Model . . . 34
4.1.1 Clustering . . . 35
4.1.2 Network Comparison . . . 36
4.1.3 Determining Matches . . . 38
4.2 Prototype . . . 39
4.2.1 Clustering . . . 39
4.2.2 Network Comparison . . . 42
4.2.3 Determining Matches . . . 42
4.3 Concluding Remarks . . . 43
4.3.1 Expectations . . . 43

5 Typed Networks Extension 45
5.1 Network Types . . . 45
5.2 Model . . . 46
5.3 Prototype . . . 49
5.4 Concluding Remarks . . . 50
5.4.1 Expectations . . . 50

6 Multiple Sources 53
6.1 Generic ER Solutions . . . 53
6.2 Model . . . 55
6.2.1 Merge Result . . . 55
6.2.2 Confidence Scores vs. Match or Non-Match . . . 58
6.2.3 Redefined Match Function . . . 59
6.2.4 Reuse of R-Swoosh . . . 59
6.3 Discussion . . . 62

7 Experiments 65
7.1 Parameters . . . 65
7.1.1 Natural vs. Normal . . . 66
7.2 Measurements . . . 67
7.2.1 Precision . . . 67
7.2.2 Recall . . . 68
7.2.3 F-Measure . . . 68
7.3 Results . . . 68
7.3.1 Pairwise Threshold and Network Threshold . . . 68
7.3.2 Weights . . . 71
7.3.3 Attributes and String Matchers . . . 72
7.3.3.1 Exclusion of String Matchers . . . 77
7.3.3.2 Best Choices . . . 77
7.3.4 Compensation for Incomplete Networks . . . 78
7.3.5 Network Types . . . 78
7.3.6 Method . . . 84
7.3.7 Beyond the Limitations of the Data Set . . . 85
7.4 Concluding Remarks . . . 88

8 Related Work **91**
8.1 Entity Resolution . . . 91
8.1.1 Supervised Approaches . . . 92
8.1.1.1 Collective Entity Resolution . . . 92
8.1.1.2 Manual Postprocessing . . . 93
8.1.2 Unsupervised Approaches . . . 94
8.1.3 Distance Metrics . . . 96
8.2 Exploiting (Social) Network Relations . . . 96
8.2.1 Aggregators . . . 97
8.3 Concluding Remarks . . . 98

9 Conclusions **99**
9.1 Recommendations . . . 100
9.2 Future Work . . . 101

Bibliography **103**

A Original Data Sources **107**

B Database Schema **111**

C Statistics on the Data Sets **113**

D More Results from the Experiments **119**
D.1 Distribution of Similarity Scores for True Matches . . . 119
D.2 Precision and Recall for Different String Matchers . . . 121

E Approximate String Matching Methods **125**
E.1 Levenshtein Distance . . . 125
E.2 Jaro Distance . . . 126
E.3 Jaro-Winkler Distance . . . 126
E.4 Jaccard . . . 127
E.5 Cosine Similarity and TF/IDF . . . 127
E.6 Q-grams . . . 128

Chapter 1

Introduction

In the last decade, the internet has become more important. It is as common as water and electricity. Activities that used to take place manually or locally are now being performed on the internet, like booking a trip for the holidays, buy games or electronics, monitor your shares or taking your business to a global level.

Because internet is becoming such an integral part of our lives even social life is taking place on it. People connect to each other on the internet, chat with family members who live far away, meet new people on community sites, discuss and review the latest gadgets and even find the love of their live by surfing on the digital highway.

By using computers and the internet for all these different activities, much more measurable data becomes available. This works in two ways: 1) businesses can better serve their customers, and 2) customers can choose among more suppliers of products and services, due to market transparency. In the first case for instance, web shops can use data mining tools to discover relations between properties of customers and their purchasing behavior. By knowing the preferences of its customers a business can better serve them. As an example for the latter case, the market for tickets has become more transparent due to the internet. This makes it possible for the customer to make a more informed decision and find the cheapest tickets or the most reliable airline.

It has been known for years that many companies have your credentials, even if you are unaware of this. Today, the internet is becoming an even bigger resource of personal data. People mainly see the benefits and fun of using this medium. For many web sites you need to complete your credentials to be able to participate in fora, save high scores of games, win gadgets etc. However, by using the internet you leave traces, whether you are aware of it or not.

Most personal data you provide on web sites are used to create a login and will not be publicly available. However, since a few years has become popular to 'profile' yourself on a web site, showing some personal information, having a blog and sharing photos and videos. In the beginning, only a few people made their own home page. Nowadays, sites have become available at which you can generate your own page, making it easier for people with less technical skills to join. Because the personal pages are organized centrally, it is possible to provide extra features. On most of these sites it is possible to build a social network,

hence these sites are called Social Networking Sites (SNSs).

Already, people saw new opportunities of social networks online. Besides, the fun it brings for people to leave short messages at each others profile sites and to stay up to date about what your friends are doing, it can be very useful. For instance, it can offer you business opportunities.

1.1 Research

This research is commissioned by Topicus FinCare B.V. The company is part of the PBT Holding and is a young and fast growing company that is looking for opportunities to conquer new markets. In cooperation with a large mail-order company, Topicus is exploring the opportunities provided by the internet.

The trend that people tend to leave personal information publicly available on the web and the social networks they build online might be a development that could be very useful. As a company, you could enrich the data you already have about your customers. This enrichment can be used for various purposes. Improved marketing is one of these purposes, as well as risk management with respect to credit granting. From this initial perspective we started this master thesis.

1.1.1 Research Goal and Scope

It takes a couple of steps to enrich existing data with data from the internet. For each customer one should search the internet for publicly available data. The data you will find on the internet can be very diverse, unstructured and even unsuitable. So this needs preprocessing. Then, you should think of a suitable way to store the enriched data.

Apart from the steps mentioned above there are two other more important aspects: 1) you must be sure that the information found on the internet does indeed belong to the same person that is represented by the customer information, and 2) privacy. Although people placed this personal information on the internet themselves, they did not do this with the intention of providing data for integration with other information. Hence, for privacy reasons, integration of personal information is quite a sensitive topic.

Although the privacy aspect of this research is quite interesting, we only address this topic from a technical perspective. We try to match profile information from one profile site with another, as shown in Figure 1.1. In the literature, this problem is referred to as Entity Resolution (ER).

In order to match profiles from different sites, a large data set from the internet is used. Suppose s_1, s_2 and s_3 are profile sites at which people have a profile page on which they share personal information. With a crawler the data on these profile pages is retrieved. This raw data needs preprocessing. For instance, based on knowledge of the structure of the profile site, content can be extracted. Then, the irrelevant data can be filtered out. It depends on the purpose what data is relevant and what data is not. After preprocessing the data sets are ready for the matching process. This part is marked with the darker rectangle in Figure 1.1. Our focus lies on determining if we may integrate two profiles, based on the probability that these two profiles belong to the same person. Calculating this probability is based on the equality of textual

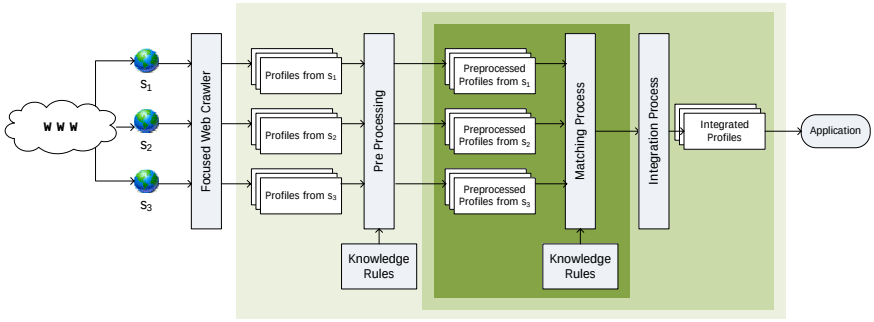


Figure 1.1: General global schema of the integration system

representations of a person. As we will see later on, this can be unreliable. We believe we can improve the reliability of calculations by involving the social networks of such profile pages.

After the matching process, the matches can be integrated and the merged information can be used for a certain application.

1.1.2 Research Questions

The research can be expressed in several research questions. The main question is:

> *Can we identify which profile pages belong to the same entity based on the textual representation of a person (the profile) and the connected profile pages (the network of a profile)?*

We can divide this question into subquestions:

- *What data is available on the different profile pages?*
- *How can you identify if two (or more) pages belong to the same entity?*
- *How can information about the network of a person on such profile sites contribute to identify a match of two profiles?*
- *Can we improve the matching process by incorporating networks?*
- *Are the results of the matching process incorporating the networks sufficiently reliable? And for what purposes?*

1.1.3 Research Approach

First, we explore the literature for solutions already used for matching problems. From the literature we extract what we think is useful.

Once we have an idea of how we can match the profile pages, we build a prototype as proof of concept. With this prototype, we will perform experiments on a data set. The results will then be analyzed, to see if our approach does indeed improve the matching process with respect to a model in which no networks are involved.

Finally, from the experimental results we will draw conclusions and give recommendations for when to use this model.

1.1.4 Contributions

In this thesis we explore entity resolution for profile pages from social network sites. Data completed at these profiles can be unreliable and hence more difficult to match with other profiles. To overcome this problem we created a model that involves social networks belonging to these profiles. By determining the amount of overlap between networks of two profiles, we can make decisions with greater certainty about the equality of these profiles.

In order to validate this statement, we built a prototype that implemented our model and retrieved our own data set to do experiments. Unfortunately, this data set has its limitations.

From the experiments it seems that the data is not so uncertain as expected. That is why entity resolution without networks involved already perform very well. Involving networks can increase Precision at cost of the Recall. Still, this might be useful in application were a high precision is required. Other applications benefit from a higher Recall and will therefore prefer not use network comparison.

1.2 Terminology

Before we start, let us discuss some terminology for clarity.

There are a lot of different Social Networking Sites (SNSs), to which we will keep referring as profile sites. We will be using the term *profile site* throughout this document. People can become a *member* of such a site and start making their own profile. We will be referring to these profiles as *profile pages*. A member is the representation of the *entity* to which this profile page belongs. A person (entity) can become a member at multiple profile sites at the same time. A visualization of these concepts and how they relate is depicted in Figure 1.2.

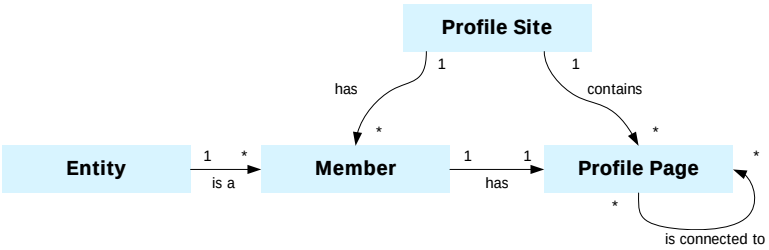


Figure 1.2: Visualization of the terms and their relationships that play an important role in this research.

1.3 Organization

In the next chapter (Chapter 2) we address the data sources we deal with and express some assumptions. In the chapters after that we will start creating a model and incrementally extend it with new ideas. For each chapter, we start with some theory, then present our model and notes on the implementation of the model in the prototype and end with a short summary and some expectations. First, we will create a model that does not involve networks at

all (Chapter 3). Next, we introduce networks (Chapter 4), followed by typed networks (Chapter 5) and finally we extend the model with multiple sources (Chapter 6). Then, we put our ideas to the test with experiments (Chapter 7). Before we present the conclusions we explore some related work to put our research in perspective (Chapter 8). And finally, we will draw conclusions and give recommendations and ideas for future research (Chapter 9).

Chapter 2

Data Sources

For the enrichment of personal information, we want to search for information we can find about this person on the web. Before we can integrate this data, we need to make sure that the information found belongs to the same entity. For this matching process we like to involve information about a person's network, hence we restrict ourselves to profile sites, because the members of such sites can build a network by connecting themselves to other members of that site.

As mentioned in the previous chapter, we do not have access to a customer database, hence we simply start at a profile site and try to enrich the data found about members of this site by information on other profile sites.

Although our focus lies on the matching process, we will briefly present some issues that we would deal with when performing the steps in front of the matching process.

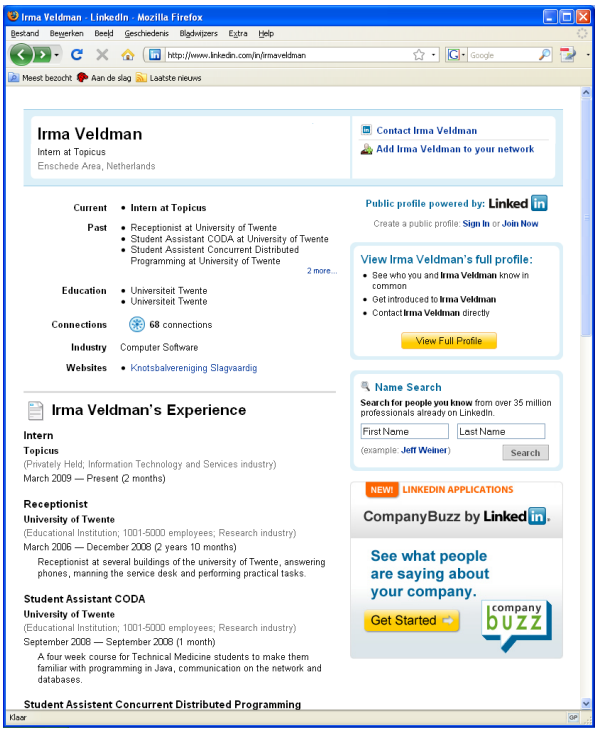
In this chapter we will further examine profile sites and discuss their properties, followed by our approach to collect a data set and its properties.

2.1 Profile Sites

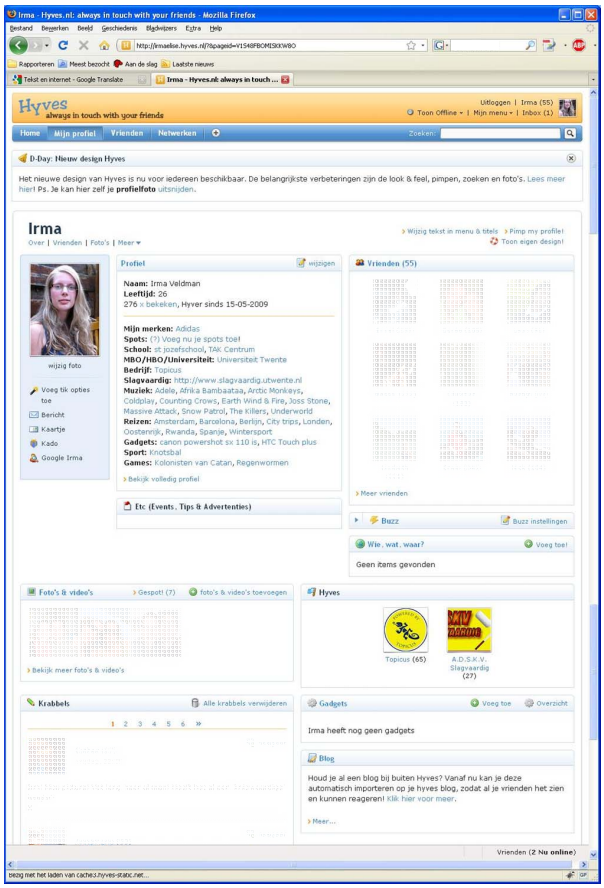
There exist many web sites where people can have their own profile. Each site has its own focus and aims for a corresponding large target group. In Table 2.1 we list a few of these profile sites and discuss their background. Notice that these sites have a huge number of members[1].

Since the purpose of these different sites differs, it is likely that a person who is active on the web, has a profile at more than one of these sites. Even for sites that do not differ much, users tend to follow their friends. If you have some

[1] Only Facebook, Hyves, LinkedIn, Netlog and Schoolbank state on their websites how many users they have. These are not all exact numbers, and can be outdated. For MySpace and Facebook we found numbers about their unique visitors on `http://www.techtree.com/India/News/Facebook_Largest_Fastest_Growing_Social_Network/551-92134-643.html`, for Facebook and Twitter on `http://blog.compete.com/2009/02/09/facebook-myspace-twitter-social-network/` and for Windows Live Spaces: `http://en.wikipedia.org/wiki/Windows_Live_Spaces`. Pierre Sauvignon did a little research himself in order to estimate the number of actual Flickr users: `http://www.thejuicycow.com/2007/08/29/everything-you-wanted-to-know-about-flickr-but-were-afraid-to-ask/`. For Twitter we found the number of unique visitors in 2008 on: `http://mashable.com/2009/01/09/twitter-growth-2008/`. We present as an indication of how many people publish information on the internet these days.



(a) LinkedIn Profile



(b) Hyves Profile

Figure 2.1: Screenshots from two different profiles belonging to the same person

Table 2.1: List of some profile sites and their background.

Name	NrOfMembers	Since	Purpose
Facebook	>200 million	2004	Connecting friends. Share stories, photos, videos and links.
Flickr	>30 million	2004	Focused on photo management and sharing
Hyves	>8 million	2004	Connect Dutch friends. Members share photos and personal information. Friends can leave notes at each others profile.
LinkedIn	>40 million	2003	Connect professionals worldwide. Members can get back in touch with classmates and colleagues again. Due to the professional character, the network is very suitable for business opportunities.
MySpace	>200 million	2003	Connecting friends and family. Share stories, photos and videos. Provides even the possibility for dating.
NetLog	>45 million	2004	Connects friends in Europe.
Schoolbank	>3 million	2000	Connect classmates from your youth, add class photos and plan reunions. Dutch site.
Twitter	>4 million	2006	To keep friends informed about what members are doing during the day.
Windows Live Spaces	>27 million	2004	Connects friends. Share stories, photos and videos. Integrates with MSN Messenger and Hotmail.

friends that are member of another site than other friends of yours, you might become member of both. Figure 2.1 shows two profile pages both belonging to the author of this report. This offers the opportunity to combine the data that can be found about one person on the web.

The table in Figure 2.2 was found on the internet[2] and shows the overlapping members of any two social networks as a percentage of the members of the social network in the blue row. For instance, 20% of MySpace members are Facebook members as well. And 64% of the Facebook members is also a MySpace member. The difference in percentages is caused by the difference in total number of users at each site. Members are expressed as the number of unique visitors. The table suggests that the period over which the unique visitors were measured is one month (September 2007). Although it is outdated, this table supports our claim that there is an overlap between the members of these profile sites and hence we can use the concept of overlap in our matching strategy.

[2]http://blog.compete.com/2007/11/12/connecting-the-social-graph-member-overlap-at-opensocial-and-facebook/, last visited 22 April 2009

Member Overlap at Social Networks

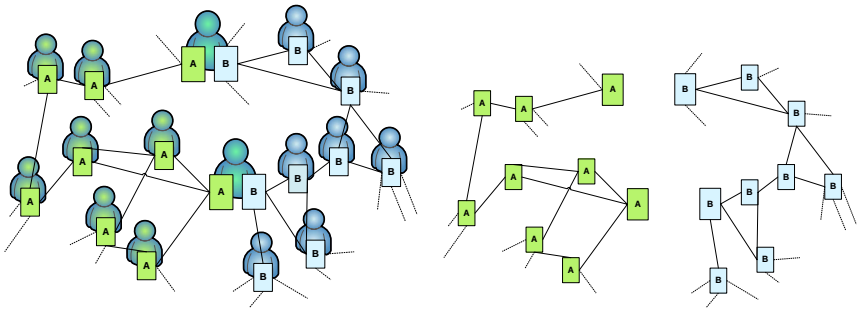
(Unique Visitors to start page/login, Sep '07)

compete.com

	Facebook	Myspace	Bebo	Friendster	Hi5	LinkedIn	Ning	Orkut	Plaxo	Salesforce	Viadeo
Facebook		64%	4%	2%	2%	2%	1%	1%	0%	0%	0%
Myspace	20%		3%	1%	1%	0%	0%	0%	0%	0%	0%
Bebo	25%	65%		2%	3%	1%	1%	0%	0%	0%	0%
Friendster	23%	49%	5%		4%	6%	2%	1%	0%	0%	0%
Hi5	24%	69%	7%	4%		1%	0%	2%	0%	0%	0%
LinkedIn	42%	32%	4%	8%	2%		8%	3%	3%	3%	0%
Ning	35%	44%	6%	6%	1%	19%		2%	2%	1%	0%
Orkut	26%	29%	3%	4%	7%	8%	2%		1%	0%	0%
Plaxo	48%	34%	5%	8%	2%	54%	14%	4%		1%	0%
Salesforce	22%	27%	4%	4%	0%	29%	3%	1%	1%		0%
Viadeo	29%	0%	1%	4%	2%	38%	10%	0%	1%	0%	

Figure 2.2: Overlap of members from different profile sites. In each row the number of overlapping members with respect to the site in the column is presented.

Figure 2.3 illustrates that in real life it is easy to find out what profile pages belong to the same person, probably because you know this person. Unfortunately, if you only have access to the profile pages without any real world knowledge, it is hard to detect if two profile pages belong to the same entity.



(a) Some persons have profiles at more than one profile site

(b) Having no real world knowledge makes it hard to integrate

Figure 2.3: A snapshot of two social networks on the internet. One with knowledge about the real world (a), in which it is easy to see which profiles belong to the same person with respect to one without this knowledge (b).

Since the information on profile pages are valuable resources for marketing, health, communication and other applications, social network analysis is gaining a new boost [1]. Social Network Analysis (SNA) has multiple fields of interests. One of them is entity resolution. We discuss this related research in the Related Work section (see Chapter 8).

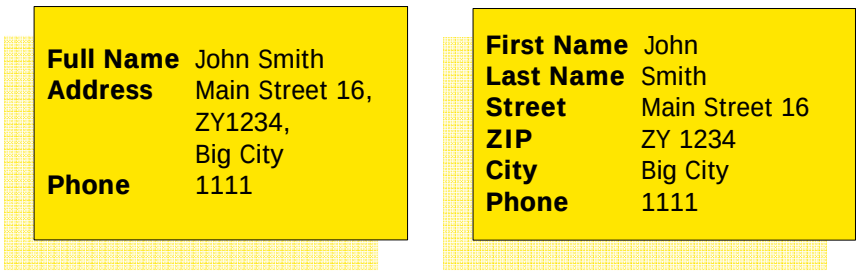
To make the problem at least a little less complex we assume that a person only has one profile page per site, as already depicted in Figure 1.2 about the terminology.

Assumption 2.1 *A person (entity) becomes a member at a profile site at most once. Hence, a person has only one profile page per profile site.*

2.2 Issues with Generic Data Retrieval

In an ideal situation, one could easily access the profile pages from all members at a central place, all uniformly structured and formatted. However, this is not the case, otherwise we would not do this research.

The data sources are heterogeneous, meaning that they are not structured and labeled in the same way. See the example in Figure 2.4. The example shows the contact record of a person named John Smith in two different address books. Notice that in Figure 2.4(a) more pieces of data are grouped (address), whereas in Figure 2.4(b) this data is split. To overcome the problem of heterogeneity we need a mapping for the different schema's in order to end up with uniformly structured records, which are desired for further processing.



(a) (b)

Figure 2.4: Contact information of John Smith in two address books with different structures

In this section we spend some words on the issues you would encounter if you want to collect data in a generic way.

Since we focus mainly on the matching process, we retrieved our data sets not in this manner. In the next section we discuss our retrieved data set and its properties.

2.2.1 Schema Mappings

Researchers have put a lot of effort in the field of schema mappings to automate the process of mapping structures from different sources automatically. A classical example is when a company wants to centralize its data storage. Data residing at different departments needs to be merged. But because these databases were developed and maintained by the different departments themselves, they cannot be merged seamlessly, due to all kind of problems, involving different schemes. Merging these storages by hand takes a lot of time and effort, hence lots of ways to automate have been thought of.

When there is a need to come with a centralized data storage, the concept of schema mappings provides the user with a unified view over all data sources. Note that it is not necessary perse to physically merge all data.

In order to provide the unified view mentioned above, there is a need for a global or mediated schema. In [23] a general definition of a data integration system is given. An integration system I consists of a global schema G, the set of data sources S and the mapping M: $I = \langle G, S, M \rangle$.

The mapping M is sound if all information in S is accessible from G, M is complete if all information in G is accessible from S and M is exact if it is both sound and complete.

This definition is kept general to apply to different kind of approaches. The approaches mostly mentioned in the literature are Global-As-View (GAV) and Local-As-View (LAV). With GAV, a global schema is created as view over all sources. This makes querying easy, since the global schema is known locally. However, it is hard to add a new source, since the global schema needs to be adapted. The opposite is LAV. There the local sources are modeled as views over a global schema. Adding a new source is easy, because it is just another view over the global schema. No adjustments to the global schema are needed. Querying however, becomes quite complex, since the only schemas known are the local views [23, 13]. GLAV (Generalized-Local-as-View) is an approach that takes features from both GAV and LAV [22].

But how can you detect automatically what the global schema should be? Several approaches are at hand and they have been classified many times as well. Leida [22] presented a short list that summarizes the approaches. We think the approaches can be presented nicely based on levels. This is shown in Figure 2.5. Per level we present an example.

- String-based. This approach measures the similarity of the string labels of an element. Measuring the similarity of strings is discussed in more detail in Section 3.1.1

  Example: Consider a source with an element `Phone Number` and a source with an element `Phone Nr`. Based on their string similarity, we might conclude these elements refer to the same attribute.

- Thesauri and Language-based. This approach is based on word representation. For instance, there can be many synonyms for one word. Moreover, in different languages, there are different words for the same things.

  Example: Consider a source having an element called `Phone` and another source with an element called `Telefoonnummer` (Dutch for telephone number). If we know that the translation of the second element is a synonym for phone, we might conclude that these elements refer to the same attribute.

- Constraint and Graph-based. This approach involves the internal representation, like the data-type in which it is presented, or the constraints defined on it. Relations with other elements are also covered in this approach.

  Example: Consider one source having an element `Phone` as an element of the relation `Person` and another source with an element `Phone` that is part of the relation `Business`. Based on this relation, we might conclude that these two elements do not refer to the same entity.

- Instance-based. This approach involves the data itself. For instance, it determines statistics about the data, or determines regular expressions.

  Example: Consider a data source for which all phone numbers of element `Phone` start with their region number followed by their subscribers' number and another source for which all phone numbers of element `Phone` start

with a prefix number indicating that these numbers belong to cell phones. Based on this data, we might decide that a resulting schema should contain both phone elements for each entity.

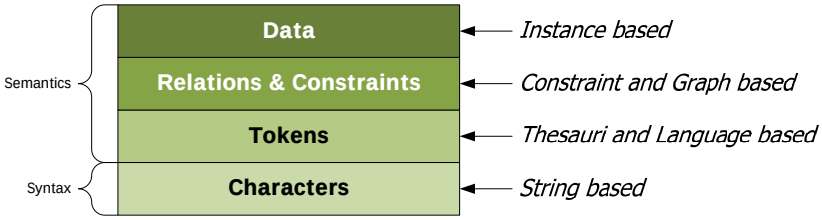


Figure 2.5: The different levels at which schema mapping approaches are performed. At each level, more and complex information is needed.

Notice that we marked the lowest level with 'syntax' and the other three levels with 'semantics'.

As mentioned in Chapter 1 we will focus on the matching of the profiles. Preprocessing the data (schema matching) is not within this scope. However, the levels mentioned above are interesting, because they also apply to the entity resolution phase, discussed in Chapter 3.

2.3 Retrieved Data Set

In order to put our matching process to the test we need a suitable data set. However, it is not possible to generate one. For generating uncertainties in the data, you need to know which uncertainties you should mimic. But the problem with uncertainties is that you'll never know exactly what you can expect. Hence, the only way to get a realistic data set is to use real data.

First, we searched for a benchmark for these kinds of researches but unfortunately they do not exist, due to privacy reasons. Thus, we had to retrieve our own data set.

2.3.1 Availability of Data

Although we have emphasized that personal data on these profile sites is freely available, retrieving it is not as straightforward as it would. For instance, for some profile pages you have to be a member yourself to be able to get access to other people's profiles. Some sites provide the members with a mechanism to protect their profile by only allowing connected people to see their profile. Hence, it is not possible to extract all member information from the internet.

A difference between traditional home pages and these profile pages is that the frequency at which they are updated is much higher [7]. Let's take Twitter: some active users post an amount of 15 messages per day. This new information could provide evidence that a decision made earlier was unjustified. However, updating the system at that rate is not an option, since it is quite costly.

Assumption 2.2 *We only take a snapshot of the information available at a certain point in time and act as if the data sources are static. We do not take updates of the data sources into account.*

In Appendix A there is an overview of which data users can share on the different data sources. A selection of interesting data is shown in Figure 2.6. We left out the most personal fields, such as 'favorite music', 'gadgets', etc. as well as fields that are unique for one profile site. We tried to keep the names of the fields as similar as possible to the original sources, hence some field names are in Dutch. The italic fields have subfields. These fields can occur more than once.

2.3.2 Data Format

The data is gathered from web pages and hence is formatted in HTML. After the retrieval step we need to store the profile data in a format that is suitable for the next steps in the process. Roughly speaking there are two candidate formats: 1) the popular data exchange format XML, or 2) the more traditional relational format. We assume that readers have some basic knowledge of XML and relational data. At [32] they have some guidelines of when to choose between a relational model and XML. We briefly describe the main differences between them and discuss the advantages and disadvantages of the use of both data formats.

Relational Model Relational data and Relational DataBase Management Systems (RDBMSs) have been extensively used over the past decades. RDBMSs today, support many features and have been optimized a lot. Many applications are built on top of an RDBMS and a lot of tooling is available.

Relational databases are very suitable to store data about related items, as the name already indicates. Normalization processes use of these relations to reduce the amount of redundant data in the database, by placing repeated data in related tables.

Relational databases are set-oriented, the data records are not stored in a certain sequence, which is also valid for the columns (or attributes).

An RDBMS can be queried by the Structured Query Language (SQL). It is a descriptive query language. SQL can be extended with stored procedures to provide extra functionality. Moreover, many RDBMSs provide support for data mining.

XML XML is a much younger technique. However, lots of efforts have been put in to improve, since it has been addressed by many people as the data exchanging format of the future. For instance, native XML databases have been developed, which have been improved by extending the functionality and optimize the performance. Unfortunately, XML databases are not as fast as RDBMSs, when it comes to highly structured data.

With XML you can also describe relations, but the power of XML is that it can present hierarchies. Another advantage of XML is its flexibility, since the schemes are not that rigid.

In contrast with relational data, XML is sequence-oriented so order matters.

We decided to work with the relational data model, since it allows us to use tools that already have functionality that comes at hand for this system. We expect that further processing will probably need OLAP (OnLine Analytic

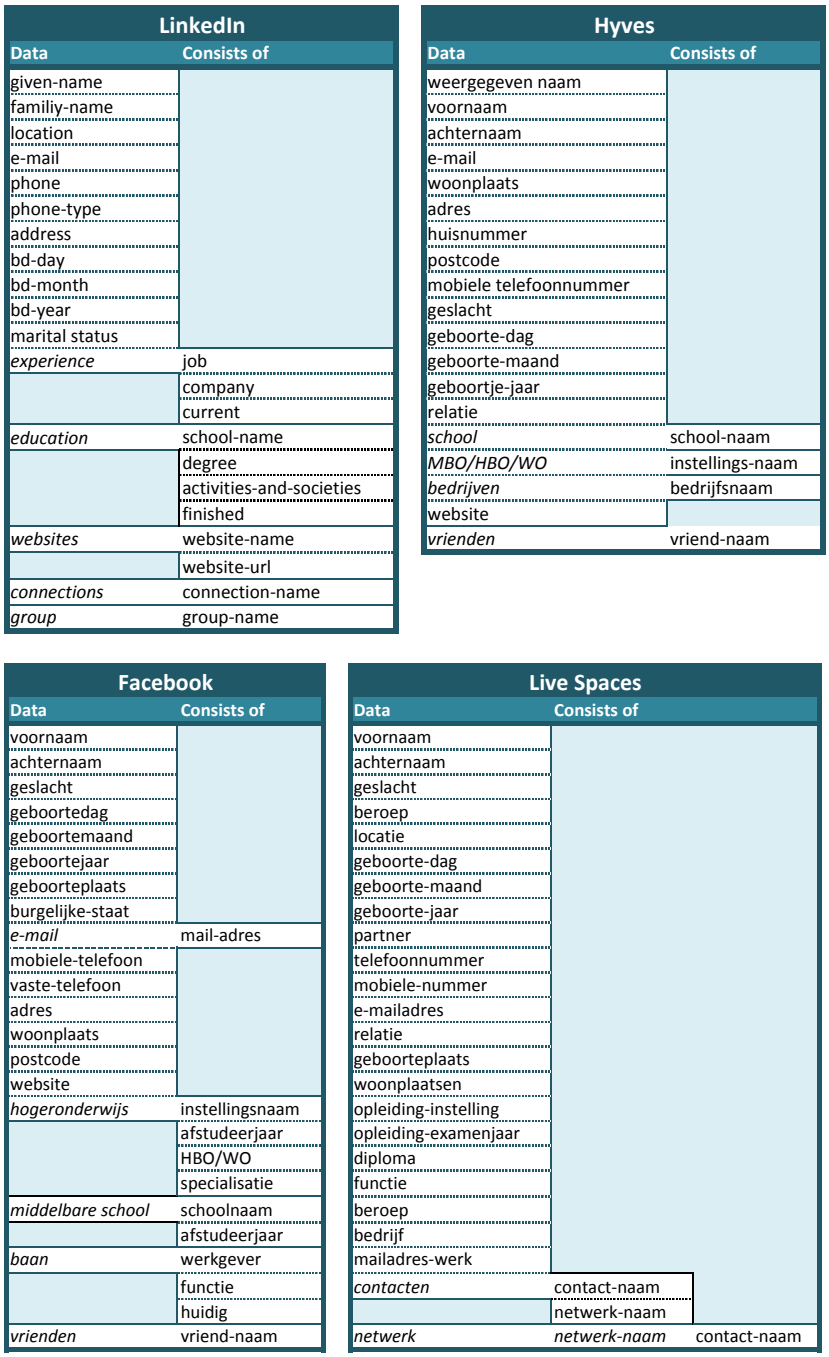


Figure 2.6: A selection of the data that a member can share on his profile page.

Processing), hence the data already being in relational format is desired. Also, we are not interested in sequence of data records, since profile sites themselves do not have a specific order for their members. Besides, we believe that the concepts of relations in the communities can be expressed easier in a relational data model.

2.3.3 Crawling the Internet

To retrieve real data we chose two profile sites to crawl. As mentioned before, we did not generate a data set, but retrieved real data from the internet. Unfortunately, to get access to enough data, we had to use a real account to view data that would be inaccessible otherwise. To be able to retrieve data from connected profiles it was necessary for the profile belonging to this account to be connected to other people's profiles. This condition (using an account belonging to a profile that has a network) restricted us to two profile sites. To enlarge the choice, we could have created a fake account on another profile site, but the chance of creating a suitably large network was minimal. Hence, we choose for LinkedIn and Hyves.

Unfortunately, the default configuration of LinkedIn is that only friends can access your list of connected profiles, which restricted us to only crawl profiles directly connected to the central profile and the profiles connected to those. We will refer to the first category as *level-1 profiles* and to the latter as *level-2 profiles*. The profile we used as a starting point will be referred to as the *level-0 profile*.

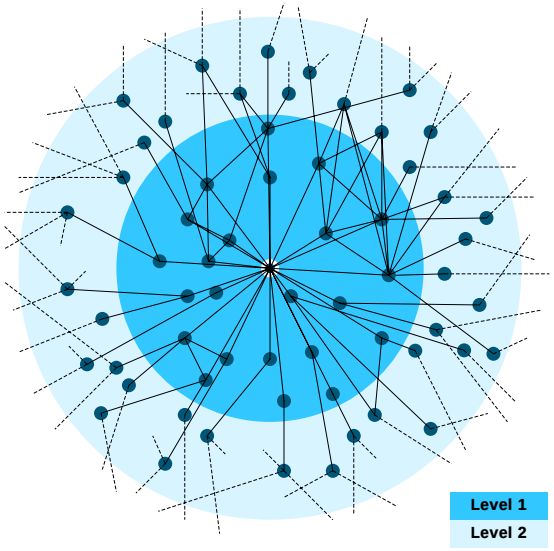


Figure 2.7: Graphical presentation of the levels. All profiles that are connected to the level-0 profile are level-1 profiles. All connections of level-1 profiles that are not level-1 profiles themselves are level-2 profiles.

This situation is depicted in Figure 2.7. The center dot is representing the level-0 profile. All profiles connected to this profile are level-1 profiles. The connections of the level-1 profiles are either level-1 profiles or level-2 profiles.

The connections of all level-2 profiles are invisible, except for those with level-1 profiles.

Hyves does not have such a default configuration, but we decided to retrieve only level-1 and level-2 profiles here as well in order to be able to manually determine the ground truth. Moreover, our data set would grow exponentially with every level, since it is said that every person is connected with every other person in the world within seven steps [2].

In Table 2.2 we show some numbers on the retrieved data sets.

Table 2.2: Some numbers on the data set. Numbers are split over level-1 profiles (L1) and level-2 profiles (L2). Level-1 matches do not include matches between a level-1 and a level-2 profile.

#Hyves P.		#LinkedIn P.		#Matches	
L1	L2	L1	L2	L1	L2
47	2133	91	2067	17	94

2.3.4 Ground Truth

Because we are handling real data from people we know, we were able to mark which profiles from these two sources belong to the same person. This is important because we can compare the outcome of the matching process with this ground truth.

The number of matches in the data set is also included in Table 2.2. A match is a level-1 match if both profiles involved in this match are level-1 profiles or a level-2 match otherwise.

2.3.5 Uncertainty of Data on Profile Pages

LinkedIn is a place where the data provided by the user is assumed mostly true, due to its formal character. However, the amount of available profile data is not very large. Almost nobody publishes their phone number. Moreover, mail addresses are only visible to connected profiles. Also, cities are presented as a region. This leaves almost only the name of a person as a good matching attribute.

For a Hyves member it is not so important to complete all data fields very reliably, due to the very informal character of Hyves. There is more data available per member, but most of that is not corresponding with data from LinkedIn, such as personal interests, favorite brands, favorite hot spots, etc. Formal information like address is seldom available. However, we were surprised to find out that names (which are probably the easiest to match) are completed properly most of the times, which makes Hyves more reliable than we thought.

The choice for LinkedIn and Hyves was made, based on the availability of both networks and the possibility to retrieve a good data set with overlapping profiles.

The final schema of the database can be found in Appendix B. Also, we performed some statistical analysis on the data set. The results can be found in the different tables in Appendix C.

Chapter 3

Simple Profile Matching

It often happens that multiple data sources need to be integrated. Schema mappings (discussed in Section 2.2.1) already handled this on schema level. Integration at instance level is called Entity Resolution (ER). It is necessary in case data sources contain data from the same domain, such as customer relation data and an overlap exists, meaning that data about one real life entity is in more than one data source. Merging these sources could lead to a duplicate data record for one entity, which violates the assumption that the data source is a representation of the real world. ER deals with the detection of duplicates in the data and processes them correctly.

A lot of research has been done on ER. It is addressed from many perspectives (e.g. databases, artificial intelligence, information retrieval) and hence there are many other names for ER (such as object identification, data cleaning, deduplicating, approximate matching, approximate joins and fuzzy matching) [35].

ER can be explained best with an example: we have two companies, company A that is a bookseller and company B, which is an electronics store. Both companies are mail order companies. They want to integrate their databases, because it could offer marketing opportunities. Of course, it is possible that a customer of company A is a customer of company B, too. It is smart to merge the information about this customer, because then you can better predict which products he or she is interested in. For instance, if a customer bought books about office tools on the computer, he might also be interested in a printer or cartridges.

In Figure 3.1 some information from the database of company A and B is shown. Let's assume that customer with `ID`='3846' of company A and customer with `ID`='2006041202' of company B are in fact the same person. Then, we encounter the following problems [16]:

- There is no universal key. Both companies create an `ID` for new customers to serve as a primary key. But because these numbers have been chosen independently, these attributes do not match. In case they do match it is not certain that they refer to the same entity.

- The way data is represented is not standardized. In this case the `Name` is a good example: 'J. Smith' versus 'Smith, J'.

	ID	Name	Address	ZIP	City	Phone
1	3845	C.Johnson	Boulevard 75	SE5468	BigCity	
2	3846	J.Smith	Mainstreet 16	ZY1234	BigCity	1111
3	3847	A. Peterson	South Avenue 1324b	AB1573	Highland	489321
4	3848	S.B.Jackson	Marshall street 17	LK5858	Millerton	6548638
5	3859	C. Johnson	Boulevard 75	SE5468	BigCity	

(a) Some customer records from company A

	ID	Name	Address	ZIP	City	Phone
1	2006041202	Smith, J	Manstreet 16	ZY1234	BigCity	2222
2	2005031401	Ray, S.T.	Merwin Road 11	PK1645	Merwin	87953
3	2006041201	Brown, G....	Beaver Dam Road 1	PL7897	Hammerto...	87953
4	2006041301	Allen, S.	Canaan Road	CV8552	Canaan	

(b) Some customer records from company B

Figure 3.1: Contact information of J. Smith in two address books with different structures

- There could be misspellings in the data. For instance, 'Manstreet' should probably be 'Mainstreet'.

- Data is not up-to-date. In the example the `Phone` numbers do not match. It could be the case that one of the phone numbers is a previous phone number.

The first problem is almost always the case when integrating heterogeneous data sources. But if we cannot use keys, we have to base our decision about two items being a match on other properties. For instance, if all attribute values are equal for both records there is a high chance that they refer to the same entity. In Section 3.1 we discuss the solutions that can check for syntactical similarity.

Unfortunately, for records to be syntactically equal is not the solution to the problem yet. For instance, assume that in the example above, the records for 'J. Smith' are exactly the same, except for their `ID`s. Based on syntax we could decide that they refer to the same entity. However, imagine a father John and his daughter Jane Smith, one being a customer at company A and the other at company B. Integrating these records is then incorrect, because the meaning of the records is different. This is a problem of semantics. In Section 3.2 we address semantic integration.

Notice that the levels that are involved in schema matching, presented in Figure 2.5, also apply in the case of ER.

3.1 Syntactical Similarity

Syntactical problems are problems that deal with language description and the representation of information at character (or substring) level. Misspellings, different word order and abbreviations belong to this category.

3.1.1 Approximate String Matching

The similarity of two strings can be measured by approximate string matching algorithms. These algorithms assign values to pairs of strings, expressing the amount of similarity. For instance, the words 'similarity' and 'dissimilarity' are likely to get a score indicating that they are very similar since they differ just three characters. The following pair of words 'similarity' and 'distance' will probably get a bad score. This score depends on the algorithm that is applied.

There are some basic concepts in string matching. Each concept has its own variations. We will briefly discuss the basic concepts and some variations below. A more detailed description of some approximate string matching methods is given in Appendix E.

We mentioned that syntactical similarity belongs to the lowest level of the diagram in Figure 2.5, i.e. the character-based level. Some of the approximate string matching methods discussed below are actually token-based, but on a syntactical level.

3.1.1.1 Character-based Methods

One of the character based methods is the Levenshtein distance (or edit-distance) [24]. This method measures the minimal number of insertions, deletions or substitutions that are needed to transform a string σ_1 into a string σ_2.

The Jaro distance [21] is expressed in the number of matching characters and the number of swapped letters. The JaroWinkler distance [35] is a variant and has a higher weighing factor for prefixes.

There are many more variants possible. Take, for instance, matchers that assign penalties to mismatched characters or bonus points to matched characters. Each variant favors some kind of type errors or performs best on a particular range of length. One variant gives less penalty for type errors that occur often (like typing an 'm' instead of an 'n' and vice versa) and less penalty for the lack or presence of accents on letters (as for 'ç' and 'c') [27].

Character-based methods can be applied also to strings that consist of more than one token. This string will be treated as if it is one token, i.e. a space-character is just seen as a normal character. This comes with some problems that token-based methods can solve (e.g. word order).

3.1.1.2 Token-based methods

Token-based methods measure the number of matching tokens between two sets of tokens. The Jaccard measure [20] is the simplest example. It measures the ratio of equal tokens in the union of tokens of both strings. Disadvantage of this method is that every word has equal weight.

TFIDF (Term Frequency / Inverted Document Frequency) [11] is a method that comes from the field of information retrieval. This method measures the frequency of a term but also corrects this with the importance of the token. This means that common tokens like 'a', 'the' and 'but' get a lower score because they are not discriminative enough.

The cosine similarity [11] expresses the different strings as term vectors, with each word being a dimension in the vector, counting the frequency of this word. The cosine similarity then measures the angle between the vectors, which is a measure for the similarity between the strings.

Unfortunately, these methods do not take misspellings into account, which means that misspellings can wrongly decrease the similarity score. Hybrid methods are able to benefit from the both the token-based methods as well as the character-based methods.

3.1.1.3 Hybrid methods

Bigrams, or, more general, Q-grams [18] can overcome the problem of ignored misspellings by dividing each token in every possible sequence of characters of length q. Now, only the tokens containing the misspellings are being rejected as match. On this new set of tokens, other token-based methods can be applied.

There is another hybrid method. The authors of [12] found out that Soft-TFIDF (hybrid TFIDF) with JaroWinkler performed best on their data sets. The token-based method will calculate a score based on the number of similar tokens and the character-based method determines if two tokens are similar (enough).

3.1.2 Examples

To see how these algorithms perform, we have tested them on several strings. For the comparison of these strings σ_1 and σ_2, we used the open-source package 'SimMetrics'[1] available at SourcForge[2], in which most of the best known approximate string matching algorithms are implemented in .NET [12].

We used the normalized scores, which means that every method $\mathsf{asm}(\sigma_1, \sigma_2) \in [0, 1]$, where $\mathsf{asm}(\sigma_1, \sigma_2) = 1$ means a complete match and the lower the value, the less similar the two strings are. We compared a number of strings with a character-based, token-based and hybrid string matcher (Levenshtein, CosineSimilarity and Q-Grams Distance respectively). The results are shown in Table 3.1.

Table 3.1: Comparison of several strings with a character-based string matcher (Levenshtein), token-based string matcher (Cosine Similarity) and a hybrid string matcher (Q-Grams Distance).

σ_1	σ_2	Lev.	Cos.	Q-Gram
John Smith	J. Smith	0.700	0.500	0.636
John Smith	Smith, John	0.090	0.500	0.400
John Smith	Mr. J. Smith	0.500	0.408	0.462
John Smith	Jhon Simth	0.600	0.000	0.333
CompName Inc.	Comp Name Inc.	0.929	0.408	0.839
CompName Inc.	CompName Incorporated	0.517	0.500	0.632

As we can see, the character-based string matcher yields to higher scores if only a minor number of characters are wrong, whereas with a token-based string matcher the complete word is seen as wrong, hence the lower score. Differences in word order are better served with a token-based handler. Hybrid string matchers return scores somewhere in between. But as we can see in the last but one row in the table, Q-Grams distance can handle concatenations/splits of words better.

[1] SimMetrics: `http://www.dcs.shef.ac.uk/~sam/simmetrics.html`
[2] `http://sourceforge.net/projects/simmetrics/`

3.2 Semantical Similarity

As already pointed out, it is not sufficient to check for syntactical similarity alone. Some instances that are syntactically equal could still belong to different real life entities. The other way around is also possible: two instances can have very low scores on syntactical similarity, but still belong to the same real life entity.

We have seen with schema mappings in Section 2.2.1 that semantical similarity can be divided in three layers. The first layer deals with synonyms for words or translations of that word to another language. The second layer deals with relationships and constraints and the last one deals with meta-data. However, ER differs from schema mappings in that we are already at instance-level, hence the third level does not apply here. We will give some examples.

Consider a person who a Dutch profile page on which he states that he likes 'voetbal' and an English page that states that he likes 'soccer'. Although the words are different, they mean the same. This example belongs to the token-level. To detect these kinds of similarity, thesauri or dictionaries can be used to look up synonyms.

For an example on the relational level, consider a person who says he likes 'soccer' on one page and 'sports' on the other. Although the words do not mean the same, they are related.

3.2.1 Ontologies

Lately, ontologies are proposed to provide a solution for semantical issues. Many definitions for ontologies are available online, but within the context of this research the definition on Wikipedia[3] is the clearest: "An ontology [...] is a formal representation of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to define the domain." E.g. ontologies facilitate a shared understanding of concepts.

The definition does not say anything about the level at which the ontology is modeled. Ontologies can simply be a glossary or thesaurus, or First Order Logic (FOL) in a more complex variant.

In [26] the author shows a picture[4] that illustrates the increasing complexity for the different levels of ontologies. We show this picture in Figure 3.2 because it is interesting to see that this picture is quite similar to Figure 2.5.

Figure 3.3 illustrates a visualization of a simple ontology. It shows some concepts (classes and instances) and relations.

Ontologies are very helpful for schema mappings. In cases where the data sources are all within a different domain, local ontologies can be defined and mapped to each other (LAV) or to a global ontology (GAV). This is a multi-ontology, respectively a hybrid ontology approach [13, 37]. Besides, there is the single ontology approach, which is more suitable in this case. In this approach an ontology is only defined for the global schema. This is no problem since the sources have the same domain, which is about personal information.

[3] `http://en.wikipedia.org/wiki/Ontologies_(computer_science)`, last time visited: 15 April 2009

[4] The picture in the work of Mika originates actually from the work of Smith and Walty (2001), but it is adjusted to fit the current state of the art.

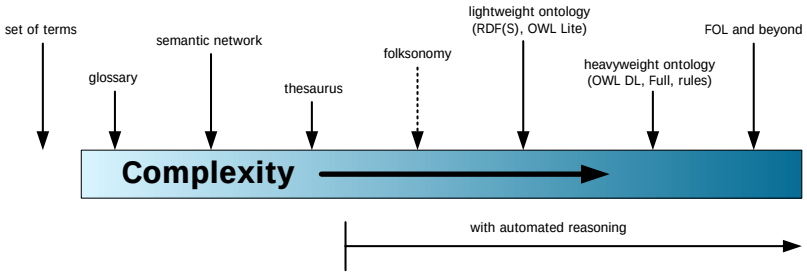


Figure 3.2: The different ontologies and their increasing complexity.

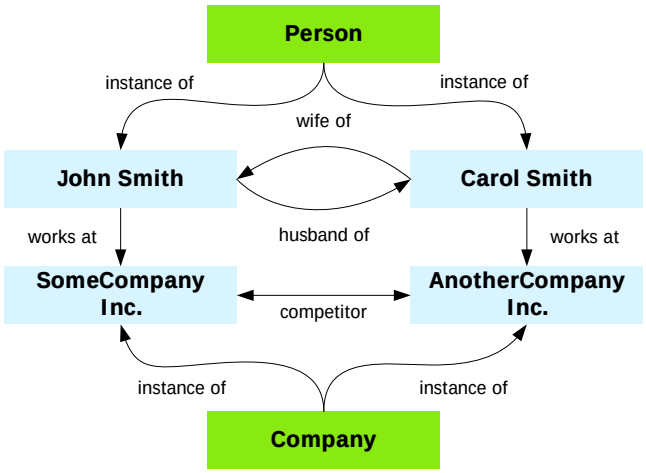


Figure 3.3: A visualization of some classes {Person, Company}, their instances {John Smith, Carol Smith, SomeCompany Inc., AnotherCompany Inc.} and relations between them (arrows).

3.3 Profile Matching Model

Since we like to know if the matching process can benefit from the concept of overlapping networks, we like to keep the model, of which the networking model will be an extension, as simple as possible. In this section we will present the simple profile matching model. In the next section we briefly discuss how this is implemented.

There are several ways to find the first indication that two profiles might match. In [34] they listed the following options: 1) pairwise vs. clustered, 2) batch vs. search and 3) effective vs. efficient.

Pairwise vs. Clustered Both pairwise and clustered duplicate detection perform on a single source that contains duplicates. Multiple representations of the same entity can occur.

With pairwise duplicate detection, all records are compared pairwise first. The pairs that are matches will then be clustered, to group all representations of one entity into one cluster. With clustered duplicate detection however all duplicate records are grouped into a cluster directly, without the pairwise com-

parison first.

In our case, the profiles should reside in their original tables before we start to match records. We do this because we assume that an entity can only have one profile per profile site (see Assumption 2.1). Therefore, we cannot directly apply the pairwise or clustered duplicate detection mentioned above. Between two sources we perform the pairwise variant. However, we do not make the decision yet based on each pair in isolation. We base our decision on other pairs as well. We will explain this later.

Batch vs. Search The second option is about the amount of entities that are involved in the process. If the ER needs to be performed for all profiles, it can be seen as a batch process, whereas a search process could perform the ER online, e.g. at query time.

Apart from the fact that we do not have access to the database with the customer information which we would like to enrich, the following considerations were made. First, with a search approach, we could discard all but a few candidate profiles. This would make the task of the ER less complex. However, to be able to determine how well our approach of ER works, we need all profiles to be involved in the process; hence a batch approach is a better choice.

Efficient vs. Effective Efficient methods aim at reducing the number of comparisons, since comparisons are very costly. Effective methods aim at a better accuracy of the matching process.

Despite the fact that efficiency is also very important, we first aim at effectiveness. We would like to find out first if incorporating the networks of profiles is helpful before focusing on efficiency.

We start our elaboration on the model with some basic definitions on profiles sites, profile pages and entities.

Definition 3.1 *S is the set of all profile sites: $S = \{s_1, s_2, \ldots, s_n\}$, P is the set of all profiles $P = \{p_1, p_2, \ldots, p_n\}$ and E is the set of real life entities $E = \{e_1, e_2, \ldots, e_m\}$.*

A profile belongs to exactly one site, denoted as $S(p_i)$, with $S(p_i) \in S$. The set of all profiles belonging to a profile site s_i is denoted as $P^{s_i} = \{p_j \in P \mid S(p_j) = s_i\}$.

In Section 2.1 we already assumed that each entity has at most one profile page per site (Assumption 2.1). We will define this more formally.

Definition 3.2 *A profile belongs to exactly one entity: $E(p_i)$, with $E(p_i) \in E$. The set of profiles owned by e_k is denoted as $P(e_k)$ with $P(e_k) \subseteq P$. However, each entity can have at most one profile at a certain profile site s_l, i.e. $\forall p_i, p_j \in P(e_k) : S(p_i) \neq S(p_j)$.*

With the relations between profiles pages, sites and entities defined, we can formally define what a pair is.

Definition 3.3 *D is de set of pairs of profiles, $D \subseteq P \times P$ and $D = \{(p_i, p_j) \mid S(p_i) \neq S(p_j)\}$.*

To make things clear, let's take a look at an example. We start with two sets of all profiles of profile sites X and Y: P^X and P^Y, depicted in Figure 3.4.

P^X	P^Y
A	I
B	J
C	K
D	L
E	M
F	N
G	O
H	P
	Q
	R
	S

Figure 3.4: Examples of two sets of profiles.

We create the set D ($P^X \times P^Y$) by taking the Cartesian product of these sets. The first part of this set is shown in Figure 3.5. Each pair consists of references to both profiles, a field to store the pairwise similarity score for this pair (denoted as $d_{pw}(p_i, p_j)$) and a field to mark whether this pair is a match or not. Note that the value for d_{pw} initially is zero for every pair.

D	p_i	p_j	$d_{pw}(p_i,p_j)$	Match
	A	I	0.0	-
	A	J	0.0	-
	A	K	0.0	-
	A	L	0.0	-
	A	M	0.0	-
	A	N	0.0	-
	A	O	0.0	-
	A	P	0.0	-
	A	Q	0.0	-
	A	R	0.0	-
	A	S	0.0	-
	B	I	0.0	-
	B	J	0.0	-

Figure 3.5: The Cartesian product of the sets P_X and P_Y from Figure 3.4

Definition 3.4 *The pairwise similarity score $d_{pw}(p_i, p_j)$ is a number in the range $[0, 1]$ that reflects to which extent the profiles p_i and p_j are alike, whereby 1 denotes complete similarity and 0 complete dissimilarity.*

How this pairwise similarity score will be calculated exactly is not specified in the definition. We did this on purpose, to keep possibilities open. The most likely option is:

$$d_{pw}(p_i, p_j) = \mathsf{asm}(p_i.attr, p_j.attr). \tag{3.1}$$

In this case $\mathsf{asm}(p_i.attr, p_j.attr)$ is a comparison function of an approximate string matcher. This function will compare the profiles p_i and p_j based on the specified $attr$, denoted as $p_i.attr$ and $p_j.attr$.

Now for every pair in D, the pairwise similarity score will be calculated. Not every pair will score very high. Pairs with a very low pairwise similarity score

are likely to be a non-match and can be discarded. For this, we introduce a threshold.

Definition 3.5 *$\tau_{pw} \in [0, 1]$ is a threshold for the pairwise similarity score d_{pw}. Each pair in D with $d_{pw} < \tau_{pw}$ is considered a non-match.*

In case of our example, τ_{pw} is set to 0.70. In Figure 3.6 we see that the pairwise similarity scores are filled in the set D. Some values do not satisfy τ_{pw} and their pairs are eliminated. The result of this elimination is a set of candidate matches, depicted on the righthand side.

D

p_i	p_j	$d_{pw}(p_i,p_j)$	Match
A	I	0.80	-
A	J	0.91	-
A	K	0.65	-
A	L	0.75	-
A	M	0.53	-
A	N	0.55	-
A	O	0.40	-
A	P	0.29	-
A	Q	0.62	-
A	R	0.16	-
A	S	0.34	-
B	I	0.83	-
B	J	0.41	-

D^{cm}

p_i	p_j	$d_{pw}(p_i,p_j)$	Match
A	I	0.80	-
A	J	0.91	-
A	L	0.75	-
B	I	0.83	-
B	K	0.71	-
C	M	0.95	-
D	N	0.87	-
E	M	0.73	-
E	O	0.73	-
F	P	0.82	-
G	Q	0.77	-
G	R	0.86	-
G	S	0.90	-
H	P	0.70	-

Figure 3.6: For each pair, the pairwise similarity score is determined. Scores that do not satisfy threshold τ_{pw} will be eliminated. This turns the set D into D^{cm}.

Definition 3.6 *D^{cm} is the set of candidate matches: $D^{cm} \subseteq D$ and $D^{cm} = \{(p_i, p_j) \in D \mid d_{pw}(p_i, p_j) \geq \tau_{pw}\}$*

From this list of candidate matches D^{cm} we need to decide which pairs are a match. We will base our matches on the highest pairwise similarity scores. Therefore, we sort the list on this score, as can be seen in Figure 3.7. The pair with the highest score is then automatically picked as being a match. In case of the example, this is the pair (C,M). As a consequence of this decision, all pairs in D^{cm} that include either profile C or M, should be eliminated, due to Assumption 2.1, in this case pair (E,M).

Finally, the list of resulting matches is depicted in Figure 3.8.

Definition 3.7 *From a list of candidate matches D^{cm} we can determine the matches as follows: first we sort the list on the highest score. Then, until the list is empty, we pick the candidate match with the highest score, let's say (p_i, p_j), mark it as a match and remove this pair and all candidate matches (p_k, p_l) from the list for which $p_k = p_i \vee p_l = p_j$ is true.*

Definition 3.8 *Each pair (p_i, p_j) upon which is decided that it is a match, is notated as $p_i \approx p_j$.*

D^{match} is the set of matches: $D^{match} = \{(p_i, p_j) \in D^{cm} \mid p_i \approx p_j\}$.

D^{cm}	p_i	p_j	$d_{pw}(p_i,p_j)$	Match
C	M	0.95	✓	
A	J	0.91	-	
G	S	0.90	-	
D	N	0.87	-	
G	R	0.86	-	
B	I	0.83	-	
F	P	0.82	-	
A	I	0.80	-	
G	Q	0.77	-	
A	L	0.75	-	
E	M	0.73	✗	
E	O	0.73	-	
B	K	0.71	-	
H	P	0.70	-	

Figure 3.7: The candidate matches D^{cm} are sorted on their pairwise similarity score d_{pw}. The first match is marked and as the consequence, another pair is eliminated.

D^{match}	p_i	p_j	$d_{pw}(p_i,p_j)$	Match
C	M	0.95	✓	
A	J	0.91	✓	
G	S	0.90	✓	
D	N	0.87	✓	
B	I	0.83	✓	
F	P	0.82	✓	
E	O	0.73	✓	

Figure 3.8: The resulting set of matches D^{match}.

3.4 Prototype

In this section we briefly describe the implementation of the model in the prototype and elaborate on the most interesting parts. In Section 7.1 we show which parameters are configurable and show what influence each of them has on the results.

We start with describing the prototype. In Listing 3.1 the simple matching process is presented in pseudo code.

Listing 3.1: The simple matching process in pseudo code.

```
Input: a set HP of Hyves profiles,
       a set LP of LinkedIn profiles
Output: a set M of matches

CM ← pairwiseCompare(HP,LP) /* set of candidate matches */
M ← determineMatches(CM)
return M
```

3.4.1 Pairwise Comparison

The matching process starts with importing the profiles from the database. Then, the first phase of matching consists of a pairwise comparison of each two profiles. Comparing two profiles will result in a similarity score d_{pw}. If this score satisfies the predefined pairwise threshold τ_{pw} this pair will end up in the list of candidate matches that is returned.

In Listing 3.2 the pairwise comparison is presented in pseudo code.

Listing 3.2: Pairwise Comparison in pseudo code.

```
Input: a set HP of Hyves profiles,
       a set LP of LinkedIn profiles,
       a stringMatcher stringMatcher,
       a set attributesToCompare of attributes,
       a double τ_pw
Output: a set CM of pairs /* candidate matches */
Require: 0 ≤ τ_pw ≤ 1

for each hp ∈ HP
    for each lp ∈ LP
        pair ← Compare(hp, lp, stringMatcher, attributesToCompare)
        if d_pw(pair) ≥ τ_pw
            add pair to CM
        endif
    endfor
endfor
return CM
```

Besides the list of profiles, this algorithm requires another two things: a stringMatcher to perform the actual comparison and a set attributesToCompare.

Which stringMatcher and which attributesToCompare will be used are defined on beforehand.

We use the stringMatchers available in a package called SimMetrics[5], an open source library for string metrics. It provides a variety of comparison methods and scores resulting from the different methods are normalized, which makes it easy to compare the scores.

The attributesToCompare contains the configurations on what attributes from which profile the comparison should be applied.

3.4.2 Determining Matches

Finally, if we have the set of candidate matches, we need to determine which of them are matches. In Listing 3.3 we present how this is done in pseudo code.

Listing 3.3: Determining matches from all candidate matches in pseudo code.

```
Input: a set CM of pairs /* set of candidate matches */
Output: a set M of matches
Require: CM ≠ ∅

OCM ← sortOnPairwiseScore(CM) /* ordered set of candidate matches */
while OCM ≠ ∅
    pair ← removeFirst(OCM)
    add pair to M
    remove all pairs from OCM that contain profiles included in pair
endwhile
return M
```

3.5 Concluding Remarks

In this chapter we presented our simple profile matching model. First, we explored the issues concerning Entity Resolution (ER) in current research. Matching records is more than comparing strings, but involves knowledge about the domain and semantics of the data. This makes it difficult to perform ER completely without human intervention.

For our model, since we will extend it with networks in the following chapters, we keep the pairwise comparison of records quite simple, i.e. we do not involve ontologies or other advanced technologies in the process.

For two sources of profiles, we will compare all profiles from one source against all profiles from the other source. This comparison returns a certain score for each pair of profiles. Pairs with a lower score than a certain threshold are excluded from further processing. For the other pairs, the so-called candidate matches, we determine the matches based on this score and the choices for other pairs.

[5] `http://www.dcs.shef.ac.uk/~sam/stringmetrics.html`, last visited June 17, 2009

3.5.1 Expectations

We believe that there are quite some discrepancies between the shared by users at different profiles. Therefore, we expect that the pairwise threshold τ_{pw} should not be very close to the value 1, since we would miss too many matches. But lowering the value for the threshold means that the list of candidate matches grows, which has a bad effect on the Precision of the matching process.

With respect to the attributes and string matchers, we expect that attributes such as Name and Email give good results with character-based string matchers. Attributes such as Schools and Companies are more sensitive for different textual representations (with abbreviations, including the city, etc.). Hence, we think these perform better with token-based string matchers.

In Chapter 7 we test these expectations in the experiments.

Chapter 4

Profile Matching with Social Network Support

We already discussed that a network is an important part of a profile page. You can invite other members of the same profile site to be your friend or, more generally, your connection. To which other members a person is connected might give us useful information during the matching process. This is depicted in Figure 4.1.

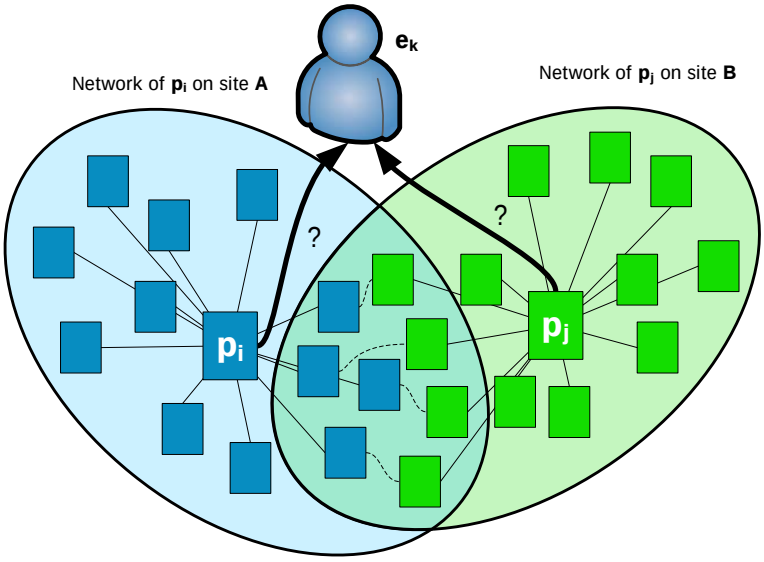


Figure 4.1: The networks belonging to the profiles of p_i on site A and of p_j on site B. The question is whether p_i and p_j belong to the same real life entity e_k. The networks of p_i and p_j might provide useful information.

In this picture p_i and p_j present profiles at a different profile sites. The question is whether both p_i and p_j belong to the same real life entity e_k. Let's assume that the data on the profiles already indicate that this might be the

case. The networks of p_i and p_j could give us the support for this decision. In the picture this support is represented as the overlap of the two ellipses. This overlap means that a person to whom p_i is connected has also a profile at the other page and is a connection of p_j, too. Notice that deciding whether a connection of p_i is also a connection of p_j is in itself the same problem as deciding whether p_i belongs to the same entity as p_j.

The concept of multi-relational integration is not new. The often used case in the literature is about integrating or de-duplicating reference systems. For instance, checking if the references from two publicly available search engines correspond to each other.

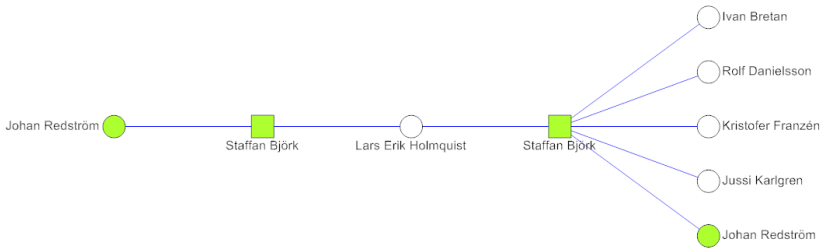


Figure 4.2: Screenshot of D-Dupe. The authors associated with the rectangles are the matching candidates. The circles represent their co-authors. Circles in between the candidates are the shared co-authors.

The authors of [6] used the reference systems case to see if they could take the co-author relationship into account.In Figure 4.2 a screenshot of their tool (D-Dupe) is shown. The source apparently contains two authors named 'Staffan Björk', but with a different `id`. If we conclude that these names refer to the same author, then this author has a list of co-authors that contains an author named 'Johan Redström' twice. The merging of the two 'Staffan Björk' author records gives us a stronger indication that both 'Johan Redström' author records also refer to the same entity.

Notice that in the work of [6] the merging of two records implies that there is stronger evidence that two co-authors in fact belong to one entity. This is different for our approach. We turn this reasoning around. If there is an indication that two profiles belong to the same entity, the overlap of their network provides stronger evidence that the profiles do indeed belong to the same entity. This will improve the results of the matching process.

Another reason to turn the reasoning around is because of the nature of the case. Overlap in social networks happens all the time, also in real life. Two persons with overlapping networks are not necessarily the same person. Hence, we only take the networks into account, if we already have some indication that the two profiles might belong to the same entity.

4.1 Model

In order to extent the simple model with the notion of social networks we will first formally define connections and networks.

Definition 4.1 *A profile can connect to other profiles. Let R be the set of all*

connections, then $R \subseteq P \times P$ *and* $R = \{(p_i, p_j) | i \neq j, S(p_i) = S(p_j)\}$. *The set of connections* R *is symmetric, i.e. if* $(p_i, p_j) \in R$ *then also* $(p_j, p_i) \in R$.

Definition 4.2 *The collection of connected profiles of a certain profile* p_i *is defined as the network of* p_i, *i.e.* N_{p_i}. *Such a network is a subset of* P, $N_{p_i} \subseteq P$ *and is defined as* $N_{p_i} = \{p_j | (p_i, p_j) \in R\}$.

As mentioned in the previous section, before we start comparing the networks of two profiles, we require an indication that these profiles refer to the same person. Hence, the first step of the process, the pairwise comparison, will remain the same. After this step we will include some extra steps to be able to base the choice whether or not a pair is a match on both the pairwise score and the overlap in networks.

We will continue with the example from the previous chapter. Suppose we have finished the pairwise comparison, then the result is the list of candidate matches shown on the righthand side of Figure 3.6. Only this time, a pair has an extra field to store a network similarity score, see Figure 4.3 for an example.

p_i	p_j	$d_{pw}(p_i,p_j)$	$d_{nw}(p_i,p_j)$	Match
A	I	0.0	0.0	-

Figure 4.3: A pair in the profile matching process with social network support contains an extra field to store the network similarity score d_{nw}.

Before we elaborate on determining the overlap between the networks of two profiles, we will first explain clustering.

4.1.1 Clustering

With clustering we divide the list of candidate matches in clusters. The goal of clustering is to divide the process into smaller independent pieces of work, which will reduce the number of comparisons. On each cluster we can perform the rest of the process without this having effect on any other cluster. We can assure this independence of clusters by making sure that all pairs in which a certain profile p_i is involved will end up in one cluster.

Let's resume our example from the previous chapter after the pairwise comparison. We ended up with the set D^{cm} shown in Figure 4.4.

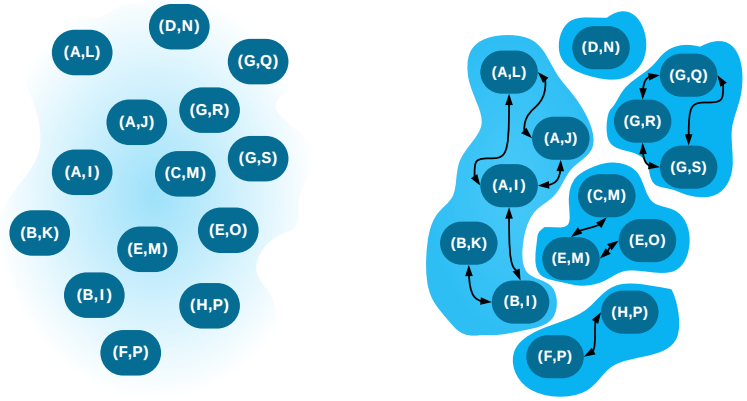
Suppose each pair is represented by a node. For each two pairs (p_i, p_j) and (p'_i, p'_j), if $p_i = p'_i$ or $p_j = p'_j$, then we can draw an edge between these pairs. This is depicted in Figure 4.5. Each strongly connected subgraph represents a cluster.

Definition 4.3 *From all candidate matching pairs* (p_i, p_j) *we can build a graph* $G = (V, L)$. *The set of vertices* V *is presented by the set of candidate matches* D^{match} *and for each two pairs* $(p_i, p_j), (p_k, p_l) \in D^{cm}$ *that share a profile, i.e.* $i = k \vee j = l$, *there is a link* $l \in L$.

A cluster of candidate matching pairs consists of pairs of profile pages (p_i, p_j) *that form a strongly connected graph.*

D^{cm}	p_i	p_j	$d_{pw}(p_i,p_j)$	$d_{nw}(p_i,p_j)$	Match
	A	I	0.80	0.0	-
	A	J	0.91	0.0	-
	A	L	0.75	0.0	-
	B	I	0.83	0.0	-
	B	K	0.71	0.0	-
	C	M	0.95	0.0	-
	D	N	0.87	0.0	-
	E	M	0.73	0.0	-
	E	O	0.73	0.0	-
	F	P	0.82	0.0	-
	G	Q	0.77	0.0	-
	G	R	0.86	0.0	-
	G	S	0.90	0.0	-
	H	P	0.70	0.0	-

Figure 4.4: Example of D^{cm} after the pairwise comparison.



(a) Set of pairs D^{cm}, not yet clustered.

(b) Set of pairs D^{cm} divided over five clusters.

Figure 4.5: Illustrated example of Definition 4.3. The set of pairs D^{cm} before (a) and after (b) clustering.

4.1.2 Network Comparison

For each cluster we can now continue the process of network comparison and determining the matches in isolation. Let's take cluster $\{(C,M),(E,M),(E,O)\}$ for example. For each pair we want to determine the overlap between the networks.

Definition 4.4 *The network similarity score $d_{nw}(p_i,p_j)$ is a number in the range $[0,1]$ that reflects to which extent the networks of profiles p_i and p_j overlap, whereby 1 denotes full overlap and 0 no overlap at all.*

$$d_{nw}(p_i,p_j) = \frac{|\mathsf{shared}(N_{p_i},N_{p_j})|}{\mathsf{min}(|N_{p_i}|,|N_{p_j}|)}. \tag{4.1}$$

The function $\mathsf{shared}(N_{p_i}, N_{p_j})$ *is then defined as*

$$\mathsf{shared}(N_{p_i}, N_{p_j}) = \{(p_k, p_l) \mid p_k \approx p_l,\ p_k \in N_{p_i},\ p_l \in N_{p_j}\}. \tag{4.2}$$

Question is whether $p_k \approx p_l$ holds. Notice that this could become a recursive problem, since the determination of these matches is what we are looking for in the first place. We decided to keep this decision as simple as possible. If necessary, one can always replace this simple decision method for a more complex but accurate one. We decide that the determination of matches in the networks of two profiles will be the same as for determining matches in the simple pairwise process. We will illustrate this with our example.

Let's assume that C, E, M and O have the following networks:

$$\begin{aligned} N_C &= \{A, B\}, \\ N_E &= \{A, B, F\}, \\ N_M &= \{I, L, S\} \text{ and} \\ N_O &= \{P, Q, R\} \end{aligned}$$

For the connected profiles of the pair (C, M) we create a list of pairs (Cartesian product). Some pairs were not contained in the D^{cm} set, because they did not satisfy τ_{pw}. We can eliminate these pairs in this list immediately, see Figure 4.6.

p_k	p_l	$d_{pw}(p_k, p_l)$	Match
A	I	0.80	-
A	L	0.75	-
A	S	-	✗
B	I	0.83	-
B	L	-	✗
B	S	-	✗

Figure 4.6: The list of pairs, created from the networks of profiles C and M. The gray pairs do not satisfy τ_{pw} and hence are marked as non-matches.

This leaves us with three pairs: (A, I), (A, L) and (B, I). In line with the determination of matches for the simple profile matching process, we sort the pairs based on their pairwise similarity score. By marking the first pair as a match, the second pair cannot be a match anymore and will be marked as non-match. Finally, the last pair is marked as a match, see Figure 4.7.

p_k	p_l	$d_{pw}(p_k, p_l)$	Match
B	I	0.83	✓
A	I	0.80	✗
A	L	0.75	✓

Figure 4.7: Only two connected pairs are marked as a match in the networks of C and M.

Since we know $\mathsf{shared}(N_C, N_M) = \{(B, I), (A, L)\}$, we can calculate the network similarity score $d_{nw}(C, M)$. By calculating this score we have two simple

choices: we divide the overlapping network by 1) the size of the largest network or 2) the size of the smallest network. In the first case, a score of 100% is not possible if the networks have unequal sizes. This is strange, because it can occur that all profiles contained in the smallest network are matched with profiles in the largest network. Hence, we believe the latter case is more suitable.

$$d_{nw}(C, M) = \frac{|\mathsf{shared}(N_C, N_M)|}{\mathsf{min}(|N_C|, |N_M|)} = \frac{2}{2} = 1 \tag{4.3}$$

We can do the same for the other two pairs in the cluster:

$$d_{nw}(E, M) = \frac{|\mathsf{shared}(N_E, N_M)|}{\mathsf{min}(|N_E|, |N_M|)} = \frac{2}{3} \approx 0.67 \tag{4.4}$$

$$d_{nw}(E, O) = \frac{|\mathsf{shared}(N_E, N_O)|}{\mathsf{min}(|N_E|, |N_O|)} = \frac{1}{3} \approx 0.33 \tag{4.5}$$

Having these values, we can fill them in in the network similarity scores d_{nw} field for each pair in the cluster. In line with the pairwise threshold, we introduce a threshold for the network similarity score as well.

Definition 4.5 *$\tau_{nw} \in [0, 1]$ is a threshold for the network similarity score d_{nw}. Each pair in a cluster with $d_{nw} < \tau_{nw}$ is considered a non-match.*

Suppose we choose our threshold as follows: $\tau_{nw} = 0.40$. The resulting cluster looks like Figure 4.8.

p_i	p_j	$d_{pw}(p_i,p_j)$	$d_{nw}(p_i,p_j)$	Match
C	M	0.95	1.00	-
E	M	0.73	0.67	-
E	O	0.73	0.33	✕

Figure 4.8: The resulting cluster after the network comparison step. Pair (E, O) is removed, because it does not satisfy τ_{nw}.

4.1.3 Determining Matches

In line with the last step in the simple profile matching process, we will finish the process by determining the matches. Back then we based our decision for a match on the pairwise similarity score, but now we will base our decision on the weighted average of the pairwise and network similarity scores.

Definition 4.6 *$\omega_{pw} \in \mathbb{N}^+$ and $\omega_{nw} \in \mathbb{N}^+$ are weights for d_{pw} and d_{nw} respectively. The weighted average of the similarity scores for a pair (p_i, p_j) is simply defined as:*

$$d(p_i, p_j) = \frac{\omega_{pw} \cdot d_{pw}(p_i, p_j) + \omega_{nw} \cdot d_{nw}(p_i, p_j)}{\omega_{pw} + \omega_{nw}} \tag{4.6}$$

.

In case of the example, it does not really matter how we distribute the weight over the pairwise or network similarity scores, since pair (C, M) has a higher score on both. Pair (C, M) will hence be marked as a match. As a consequence, (E, O) is marked as a non-match, shown in Figure 4.9.

p_i	p_j	$d_{pw}(p_i,p_j)$	$d_{nw}(p_i,p_j)$	Match
C	M	0.95	1.00	✓
E	M	0.73	0.67	✗

Figure 4.9: Pair (C, M) is marked as a match, because it has the highest similarity score. Therefore, (E, O) must be marked as a non-match.

4.2 Prototype

For the matching process that involves social networks, some adjustments need to be made to the initial prototype. The high-level process is shown in pseudo code in Listing 4.1.

Listing 4.1: The matching process in pseudo code.

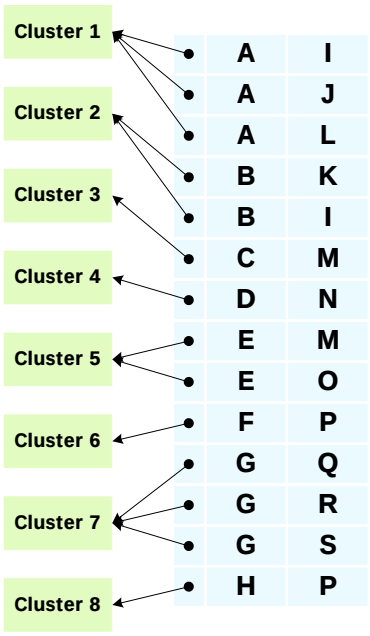
```
Input: a set HP of Hyves profiles,
       a set LP of LinkedIn profiles
Output: a set M of matches

CM ← pairwiseCompare(HP,LP) /* set of candidate matches */
CS ← cluster(CM) /* set of clusters */
for each C ∈ CS
    networkCompare(C)
    SM ← determineMatches(C) /* (sub)set of matches */
    add SM to M
endfor
return M
```

The pairwise comparison is still the same, but the clustering and the network comparison are new and determining matches is slightly changed. Therefore, we will discuss them in the next sections.

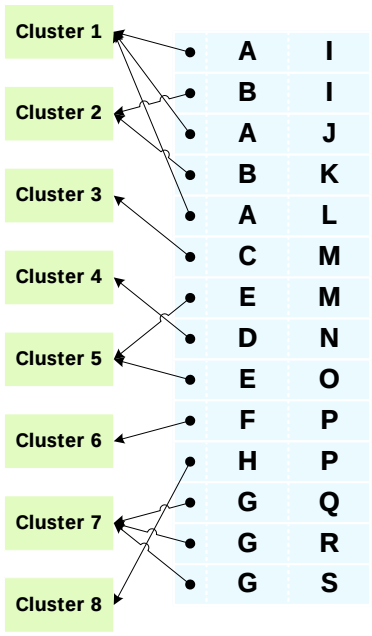
4.2.1 Clustering

We will illustrate the concept of clustering with the approach we took to implement the clustering. First, we show some steps of the algorithm graphically in Figure 4.10 and then we present the pseudo-algorithm, see Listing 4.2.

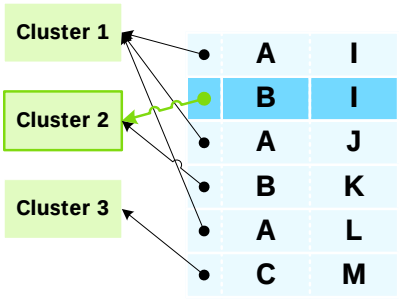
The general idea is to sort the pairs on the first profile (the profiles belonging to site X). Pairs having the same profile from X are then assigned to a cluster. This is depicted in Figure 4.10(a). Next, the pairs will be sorted on the second profile (the profiles belonging to site Y), shown in Figure 4.10(b). This time also, we need to get clusters with the same profile from Y into the same cluster. We start with the first pair and temporarily store the cluster to which it is assigned. In the example (Figure 4.10(c)), the first pair is (A, I) and it is assigned to cluster 1. Then, we take a look at the next pair: (B, I). This pair should be in the same cluster as (A, I), because they both have profile I in it. However, it is assigned to cluster 2. We will make sure that (B, I) will be assigned to cluster 2 by assigning all profiles referring to cluster 2 to cluster 1 now, see Figure 4.10(d). This is important because we already clustered these profiles,



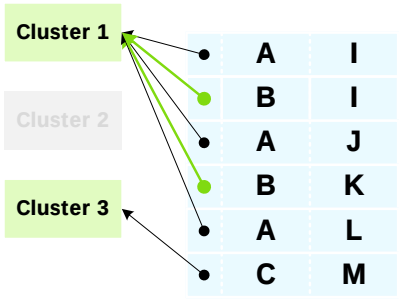
(a) The pairs are sorted on their first profile. Pairs with equal first profiles are assigned to the same cluster.



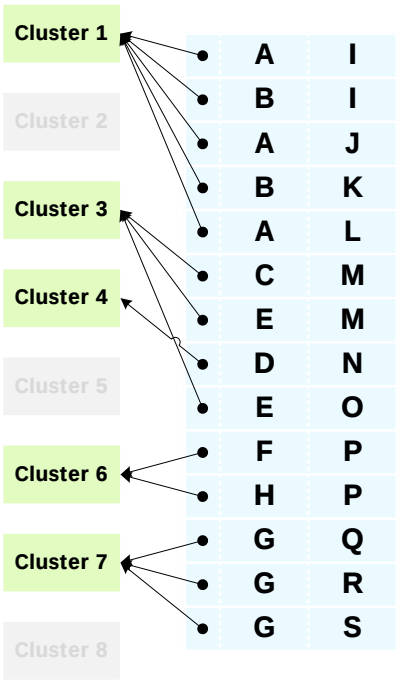
(b) During the second step of the clustering, the pairs are sorted on the second profile.



(c) Cutout showing the first 5 pairs. Since we want all occurrences of one profile in the same cluster, we need to move the reference of pair `(B,I)` to the same cluster that `(A,I)` is referring to.



(d) The pairs originally referring to cluster 2 are now referring to cluster 1, to keep satisfying the property that a cluster contains all pairs that are linked.



(e) If we keep doing this for all pairs, this is the result.

Figure 4.10: Graphical representation of the clustering algorithm.

Listing 4.2: The clustering algorithm

```
Input: a set CM of pairs /* candidate matches */
Output: a set CS of clusters

previousId ← −1;
currentId ← −1;
/* First Phase */
OCM ← sortOnFirstProfile(CM) /* Ordered set of candidate matches */
for each pair ∈ OCM
    currentId ← firstProfileIdOf(pair)
    if currentId ≠ previousId
        create newCluster
        add newCluster to CS
    endif
    add pair to newCluster
    previousId ← currentId
endfor
/* Second Phase */
OCM ← sortOnSecondProfile(CM)
previousId ← −1
previousPair = null
for each pair ∈ OCM
    currentId ← secondProfileIdOf(pair)
    if currentId = previousId
        if assignedCluster(previousPair) ≠ assignedCluster(pair)
            add all pairs of assignedCluster(pair) to assignedCluster(previousPair)
            remove assignedCluster(pair) from CS
        endif
        previousId ← currentId
        previousPair ← pair
    endif
endfor
return CS
```

because they share a profile from site s_1 and hence should stay clustered. The result of the clustering is shown in Figure 4.10(e).

4.2.2 Network Comparison

Now we have partitioned the list of candidate matches into smaller portions, we can perform a network comparison on each portion and determine which of the candidate matches should be marked as a match.

Listing 4.3: Network Comparison in pseudo code

```
Input: a set C of pairs /* a cluster */
       a double τ_nw
Require: 0 ≤ τ_nw ≤ 1
for each pair ∈ C
    CHP ← getConnectedHyvesProfiles(pair)
    CLP ← getConnectedLinkedInProfiles(pair)
    create NP /* set of networkPairs */
    for each hp ∈ CHP
        for each lp ∈ CLP
            networkPair ← getPair(hp, lp)
            if exists(networkPair) /* networkPair is already a candidate match */
                add networkPair to NP
            endif
        endfor
    endfor
    NM ← determineMatches(NP) /* set of matches in network */
    d_nw(pair) ← |NM| / min(|CHP|, |CLP|)
    if d_nw(pair) ≥ τ_nw
        remove pair from C
    endif
endfor
```

From all pairs that can be made of the connected profiles of hyvesProfile and linkedInProfile that were present in D^{cm} (and thus satisfy τ_{pw}) are candidate network matches. From these candidate network matches the matches are found with the determineMatches method, presented in Listing 3.3. The network matches are counted and then divided by the size of the smallest network. This is the network score for the pair pair.

Now every pair in the cluster has a network similarity score. This network similarity score is required for determining which pairs in the cluster are marked as a match.

4.2.3 Determining Matches

Finally, if, for each cluster, it is known which pairs satisfy τ_{pw} and τ_{nw}, we can start determining which ones should be marked as matches.

Notice that this method differs only slightly from the one presented in Listing 3.3. With this variant the pairs in the cluster are sorted based on their weighted average similarity score, instead of their pairwise similarity score.

Listing 4.4: Determining Matches in pseudo code

```
Input: a set C of pairs /* cluster */
Output: a set M of matches

OC ← sortOnWeightedAverageScore(C) /* set of ordered pairs in cluster */
while OC ≠ ∅
    pair ← removeFirst(OC)
    add pair to M
    remove pairs from OC that contain profiles included in pair
endwhile
return M
```

4.3 Concluding Remarks

In this chapter we extended our simple profile matching model with networks. We saw in the literature that variants of networks involved in entity resolution are receiving more attention recently. Since we apply the model to the matching of profiles on social network sites, we can adjust the model to this specific case.

The extension of the simple model affects the determination of networks, by deciding which candidate matches are matches based on not only the pairwise similarity score, but also on the network similarity score. For a candidate match, we compare the social networks they have and calculate the overlap. This overlap is a measure for the network similarity. Again, we introduce a threshold that needs to be satisfied.

Before we perform the step in which the network scores are determined, we group the candidate matches in clusters. For each cluster the network matching phase can be performed independently from the other clusters. Dividing the work in the network matching phase into smaller portions will increase scalability.

4.3.1 Expectations

We introduced this variant of the model because we believe involving the networks will improve the matching results. We believe that people having multiple profile pages have a certain amount of overlap in their networks. These networks provide extra support for a candidate match to be a correct match, indeed. Our expectation is that this model gives better results than the model presented in the previous chapter. It is hard to estimate what the optimal value for the network threshold should be, but we do not expect an overlap of over 50% in general. From the experiments it will appear what the optimal value for this threshold should be.

In the next chapter we introduce a more specific variant of the model involving networks.

Chapter 5

Typed Networks Extension

Networks can be very complex structures. In [28] the authors unravel some of the most interesting statistical features of these interconnected structures. As one of the examples, human social networks are mentioned. They show how the network of one person can contain several subnetworks that might overlap. Someone can have friends, colleagues and family for instance. And some of your colleagues may also be friends of yours.

When you have a profile at a profile site and start connecting with other people having such a profile on the same profile site, it happens often that you can label this connection with a type. For instance, you can say, "this person is a friend" or "we attended the same school". Labeling such a connection could provide you with some extra information that might be useful during the matching process.

Inspired by the work in [28] and the labels you can assign to your connections on those profile pages we extend our model of profile matching even more by taking into account these kinds of labels. We will elaborate on this in the next section and will then extend the model and prototype accordingly.

5.1 Network Types

We can make a distinction on the type of networks a person has, since it can give us more information about the likeliness of a match. Figure 5.1 illustrates that a person can have connections in different networks. It is possible for a connection to be in more than one network, which is depicted by the overlapping networks.

It is useful to make this distinction, which we will explain with an example. Consider two profiles for which we want to know whether they belong to the same person. Profile p_i in Figure 5.2(a) is only connected to 2 colleagues and 55 friends. Profile p_j in this situation has only 3 connections to colleagues and 67 connections to friends. We may conclude that both profiles sites are basically used for connecting friends. However, for the number of friends connected to these profiles, only a small number of those friends is contained in the network of both profiles.

In the situation of Figure 5.2(b) we can conclude that the profiles are used for different purposes. Profile p_i is used mainly for connecting with colleagues

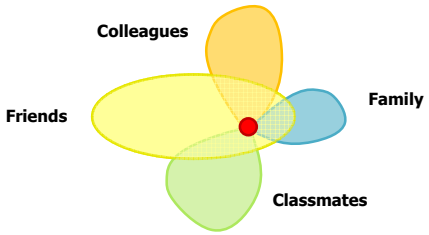


Figure 5.1: Illustration of the type of networks a person can have. A connection of this person can be in more than one network, hence the overlaps.

and p_j mainly for connecting with friends. Due to the difference in (derived) purposes the small amount of overlap is more expected in this case.

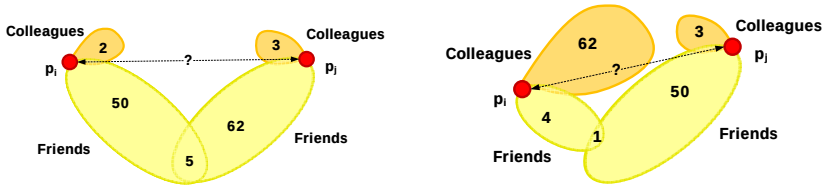


Figure 5.2: Illustration of the difference in impact of an overlap, depending on the types and sizes of the networks.

If we would like to take the notion of different networks into account, we need to define different types of connections.

5.2 Model

To extend our model with types, we need to introduce some new definitions. We start with the set of types T.

Definition 5.1 *If T is the set of types $T = \{t_1, t_2, \ldots, t_n\}$, then $T(p_i, p_j)$ denotes the set of types of a connection, in case this connection exists, i.e. $T(p_i, p_j) \subset T$.*

From this definition follows that every connection has a type. It is not always clear what the type of a connection should be, hence we introduce the type unknown. A connection can never have both the type unknown and one or more other types, hence the strict subset in Definition 5.1.

Remember the example from the previous chapter, where we introduced networks. For the candidate match (C, M) we examined their networks. We will do this again and now take the types of their networks into account. In this example, $T = \{\text{friends}, \text{colleagues}\}$.

Definition 5.2 *We define a typed network $N_{p_i}^{t_j}$ as the set of profiles that are connected to p_i and have type t_j. Thus, $N_{p_i}^{t_j} \subseteq N_{p_i}$ and $N_{p_i}^{t_j} = \{p_k \mid (p_i, p_k) \in R,\ T(p_i, p_k) \ni t_j\}$.*

Recall that for the cluster with C, E, M and O we have

$$\begin{aligned} N_C &= \{A, B\}, \\ N_E &= \{A, B, F\}, \\ N_M &= \{I, L, S\} \text{ and} \\ N_O &= \{P, Q, R\} \end{aligned}$$

We will divide these sets over the different types:

$$\begin{aligned} N_C^{\mathsf{friends}} &= \{A\}, \\ N_C^{\mathsf{colleagues}} &= \{B\}, \\ N_E^{\mathsf{friends}} &= \{A, B\}, \\ N_E^{\mathsf{colleagues}} &= \{F\}, \\ N_M^{\mathsf{friends}} &= \{I, L\}, \\ N_M^{\mathsf{colleagues}} &= \{S\}, \\ N_O^{\mathsf{friends}} &= \{P, Q, R\} \text{ and} \\ N_O^{\mathsf{colleagues}} &= \emptyset \end{aligned}$$

If we want to incorporate the types of the network, we need some adjustments to the model. We want the network similarity score d_{nw} to be a measure of the number of overlapping network profiles that belong to the same type. Below we redefine d_{nw}.

Definition 5.3 *The network similarity score for the typed networks model is as follows:*

$$d_{nw}(p_i, p_j) = \sum_{t \in T} \frac{d^t_{nw}(p_i, p_j)}{|T|} \tag{5.1}$$

It means we will calculate the network similarity score by summing the separate typed network similarity scores for all types $t \in T$, divided by the size of T.

Definition 5.4 *The typed network similarity score $d^t_{nw}(p_i, p_j)$ is defined as*

$$d^t_{nw}(p_i, p_j) = \begin{cases} \frac{|\mathsf{shared}(N^{t_m}_{p_i}, N^{t_m}_{p_j})|}{\mathsf{min}(|N^{t_m}_{p_i}|, |N^{t_m}_{p_j}|)} & \text{if } |N^{t_m}_{p_i}| > 0 \ \wedge \ |N^{t_m}_{p_j}| > 0 \\ 0 & \text{if } |N^{t_m}_{p_i}| = 0 \ \vee \ |N^{t_m}_{p_j}| = 0 \end{cases} \tag{5.2}$$

and finally, the typed equivalence of the function shared *is*

$$\begin{aligned} \mathsf{shared}(N^{t_m}_{p_i}, N^{t_m}_{p_j}) = \{(p_k, p_l) \mid\ & p_k \approx p_l \wedge \\ & p_k \in N_{p_i} \wedge \\ & p_l \in N_{p_j} \wedge \\ & t_m \in T(p_k, p_i) \ \cap \ T(p_l, p_j)\} \end{aligned} \tag{5.3}$$

In Figure 5.3 we see the result of the network comparing phase for C and M. Only three records are candidate network matches. For these candidate matches we can now decide if they match, based on their pairwise score and whether their connections with C and M have the same type.

p_k	p_l	$d_{pw}(p_k,p_l)$	Match
A	I	0.80	-
A	L	0.75	-
A	S	-	✗
B	I	0.83	-
B	L	-	✗
B	S	-	✗

Figure 5.3: The list of pairs, created from the networks of profiles C and M. The gray pairs do not satisfy τ_{pw} and hence are marked as non-matches.

For the match (B, I) the types are not equal: $T(C, B) = \{\text{colleagues}\}$ and $T(M, I) = \{\text{friends}\}$, so $T(C, B) \cap T(M, I) = \emptyset$. This match will not be counted. However, $T(C, A) \cap T(M, I) = \{\text{friends}\}$, hence this match is counted. As a consequence the last candidate match is a non-match. This is shown in Figure 5.4. Below, we calculate the typed network similarity scores.

p_k	p_l	$d_{pw}(p_k,p_l)$	Match
B	I	0.83	✗
A	I	0.80	✓
A	L	0.75	✗

Figure 5.4: The resulting cluster after the typed network comparison step. The pairs are sorted on their total score.

$$d_{nw}^{\text{friends}}(C, M) = \frac{|\text{shared}(N_C^{\text{friends}}, N_M^{\text{friends}})|}{\min(|N_C^{\text{friends}}|, |N_M^{\text{friends}}|} = \frac{1}{1} = 1,$$

$$d_{nw}^{\text{colleagues}}(C, M) = \frac{|\text{shared}(N_C^{\text{colleagues}}, N_M^{\text{colleagues}})|}{\min(|N_C^{\text{colleagues}}|, |N_M^{\text{colleagues}}|} = \frac{0}{1} = 0 \; and$$

$$d_{nw}(C, M) = \sum_{t \in T} \frac{d_{nw}^t(C, M)}{|T|} = \frac{1}{2} + \frac{0}{2} = 0.5$$

We can do the same for E and M and for E and O:

$$d_{nw}^{\text{friends}}(E,M) = \frac{|\text{shared}(N_E^{\text{friends}}, N_M^{\text{friends}})|}{\min(|N_E^{\text{friends}}|, |N_M^{\text{friends}}|} = \frac{1}{2} = 0.5,$$

$$d_{nw}^{\text{colleagues}}(E,M) = \frac{|\text{shared}(N_E^{\text{colleagues}}, N_M^{\text{colleagues}})|}{\min(|N_E^{\text{colleagues}}|, |N_M^{\text{colleagues}}|} = \frac{1}{1} = 1,$$

$$d_{nw}(E,M) = \sum_{t \in T} \frac{d_{nw}^t(E,M)}{|T|} = \frac{0.5}{2} + \frac{1}{2} = 0.75,$$

$$d_{nw}^{\text{friends}}(E,O) = \frac{|\text{shared}(N_E^{\text{friends}}, N_O^{\text{friends}})|}{\min(|N_E^{\text{friends}}|, |N_O^{\text{friends}}|} = \frac{1}{1} = 1,$$

$$d_{nw}^{\text{colleagues}}(E,O) = \frac{|\text{shared}(N_E^{\text{colleagues}}, N_O^{\text{colleagues}})|}{\min(|N_E^{\text{colleagues}}|, |N_O^{\text{colleagues}}|} = 0 \; and$$

$$d_{nw}(E,O) = \sum_{t \in T} \frac{d_{nw}^t(E,O)}{|T|} = \frac{1}{2} + \frac{0}{2} = 0.5$$

For the candidate matches in the cluster, we can now fill their network scores. This is shown in Figure 5.5.

p_i	p_j	$d_{pw}(p_i,p_j)$	$d_{nw}(p_i,p_j)$	Match
E	M	0.73	0.75	-
C	M	0.95	0.50	-
E	O	0.73	0.50	-

Figure 5.5: For this cluster, the network scores are filled.

When we will determine the matches, based on the total score with equal weight for d_{pw} and d_{nw}, pair (E, M) will be marked as a match. As a consequence, the other two pairs in this cluster cannot be chosen as a match anymore, see Figure 5.6.

p_i	p_j	$d_{pw}(p_i,p_j)$	$d_{nw}(p_i,p_j)$	Match
E	M	0.73	0.75	✓
C	M	0.95	0.50	✗
E	O	0.73	0.50	✗

Figure 5.6: From this cluster, only the first candidate match is marked to be a match. The other pairs are marked as a non-match, due to this choice.

5.3 Prototype

To implement the extension in the prototype, only minor changes are needed. The step where the overlap for both networks is determined, which is in the determineMatches() method (Listing 3.3), presented in Section 4.2.3 needs some adjustments as well as the networks comparing step, presented in Section 4.2.2.

Main adjustment to determineMatches() in Listing 5.1 with respect to the original is that there is an extra condition to meet before a pair can be marked

Listing 5.1: Determining Typed Matches in pseudo code

```
Input: a set C of pairs /* cluster */
Output: a set M of pairs /* matches */

OC ← sortOnPairwiseScore(C) /* ordered set of candidate matches */
while OC ≠ ∅
    pair ← removeFirst(OC)
    if shareType(pair)
        add pair to M
        remove pairs from OC that contain profiles included in pair
        for each type ∈ sharedTypes(pair)
            increment nrOfNetworkPairsSharingType(type) by 1
        endfor
    endif
endwhile
return M
```

as a match. Both profiles in the pair have a type that captures the types of the relation with the profile of which they are in the network. Then, if this pair is marked as a match, the nrOfNetworkPairsSharingType(type) method registers the number of times matching network profiles were of type type, which is necessary to calculate the network similarity score. This is done in the adjusted version of the networkComparison() method presented below (Listing 5.2).

After the call to determineMatches() the network score is calculated. A call to ratioOfSharedTypedConnections() returns the number of shared connections that have a specified type divided by the total (minimum) number of connections with that type, i.e. d_{nw}^{type}.

5.4 Concluding Remarks

In this chapter we introduced a variant of the model using typed networks. We were inspired by this idea because a member can (or has to) label connections when he wants to connect to another member (such as: "Friend", "Colleague", etc.).

This label (or type) provides us with some extra information about the connection and has potential of supporting the decision for a match even better. Overlap of networks does now not only depend on pairwise similarity of network profiles, but on the type of the connections as well.

Unfortunately, this type information is not (yet) available on every site. For this research, we determined the types of connections by hand. This is quite easy to perform and although false assumptions can be made about a connection, we believe this information is more complete than the information available at the profiles sites.

5.4.1 Expectations

It is difficult to foresee what the results of this variant of the model will be. The model is stricter on when to call something a network match, hence it may

Listing 5.2: Network Comparison in pseudo code

```
Input: a set C of pairs /* cluster */
       a double τ_nw
Require: 0 ≤ τ_nw ≤ 1

for each pair ∈ C
    CHP ← getConnectedHyvesProfiles(pair)
    CLP ← getConnectedLinkedInProfiles(pair)
    create NP /* set of network pairs */
    for each hp ∈ CHP
        for each lp ∈ CLP
            networkPair ← getPair(hp,lp)
            if networkPair ≠ null
                add networkPair to NP
            endif
        endfor
    endfor
    NM ← determineMatches(NP) /* set of network matches */
    for each type in T
        d_nw(pair) += ratioOfSharedTypedConnections(type)
    endfor
    d_nw(pair) /= |T|
    if d_nw(pair) < τ_nw
        remove pair from C
    endif
endfor
```

turn out that the model performs worse than the variant with simple network comparison.

In the next chapter we explore how we can extend our model with multiple sources.

Chapter 6

Multiple Sources

Until now, we only dealt with the matching process for two sources. In practice, you would like to combine all information you can find about a person to get a highly enriched profile. Therefore, we will explore how we can handle multiple sources and build a new layer on top of the existing model.

6.1 Generic ER Solutions

In research, solutions for generic ER are already present [25, 3]. These solutions concentrate on comparing records from multiple sources in an efficient manner, i.e. reduce the number of comparisons as much as possible.

We will briefly present the issues they encountered first and will reuse their ideas in our model.

The models presented in the literature use a set that is the union of all source records (called an instance I) and this set contains duplicates. They abstract from match and merge functions. As soon as the match function determines that two records are the same, the merge function will create a composed record. This newly created record is then added to the set of all records and must be compared against all other records already present in the set.

In [3] they also state that if the match function is reflexive and commutative, and the match and merge function are in the Union Class, the ICAR properties are satisfied. We will further explain this.

Reflexive A function is said to be reflexive if the input and output of the function belong to the same set. Suppose we have a set of all possible records R, then $\forall r \in R,\ \mathsf{match}(r, r) \in R$.

Commutative A function is said to be commutative if the order of the operands does not matter. Thus, $\forall r, s \in R,\ \mathsf{match}(r, s) = \mathsf{match}(s, r)$.

Union Class The match and merge functions are said to be in the Union Class if they are based on the union of values. If a record with name {John Doe} and a record with {J. Doe} are merged, then the result will be {John Doe, J. Doe}.

If the conditions mentioned above are satisfied, then the ICAR properties are satisfied also. We present them below, but will first define a notation for merged records.

Definition 6.1 *We denote the merged record of two records p_i and p_j with $\langle p_i, p_j \rangle$.*

The ICAR properties are stated below.

1. Idempotence: $\forall r, r \approx r$ and $\langle r, r \rangle = r$. A record always matches itself, and merging it with itself still yields the same record.
2. Commutativity: $\forall r_1, r_2, r_1 \approx r_2$ iff $r_2 \approx r_1$, and if $r_1 \approx r_2$, then $\langle r_1, r_2 \rangle = \langle r_2, r_1 \rangle$.
3. Associativity: $\forall r_1, r_2, r_3$ such that $\langle r_1, \langle r_2, r_3 \rangle \rangle$ and $\langle \langle r_1, r_2 \rangle, r_3 \rangle$ exist, $\langle r_1, \langle r_2, r_3 \rangle \rangle = \langle \langle r_1, r_2 \rangle, r_3 \rangle$.
4. Representativity: If $r_3 = \langle r_1, r_2 \rangle$ then for any r_4 such that $r_1 \approx r_4$, we also have $r_3 \approx r_4$.

Based on these properties and conditions, the authors of [3] have come up with the R-Swoosh algorithm that does ER in such a way that the number of comparisons of records is minimized. For clarity, we will repeat this algorithm in Listing 6.1. $ER(I)$ denotes the set of records after the ER is applied.

Listing 6.1: The R-Swoosh algorithm from [3].

```
Input: a set I of records
Output: a set I' of records, I' = ER(I)
I' ← ∅
while I ≠ ∅
    currentRecord ← a record from I
    remove currentRecord from I
    buddy ← null
    for all records r' ∈ I'
        if match(currentRecord, r')
            buddy ← r'
            break
        endif
    endfor
    if buddy = null
        add currentRecord to I'
    else
        r'' ← ⟨ currentRecord, buddy ⟩
        remove buddy from I'
        add r'' to I
    endif
endwhile
return I'
```

This algorithm simply compares each record with every other record. Records that were not compared to every other record yet, are in I. Whenever a record r_1 from I is compared to the records already in I', it moves from I to I', unless

it is matched with a record r_2 from I'. Then, r_1 and r_2 are removed from I and I' respectively and the merged record $\langle r_1, r_2 \rangle$ is then added to I, since it has not been compared to all records in I'. The algorithm terminates when I is empty.

We will illustrate this model with an example, shown in Figure 6.1. We begin the ER process with $I = \{a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4, b_5, c_1, c_2, c_3\}$ During the first iterations of R-Swoosh, records a_1, a_2, a_3 and a_4 are compared against the records in I' but are not matched, so they are moved from I to I' (Figure 6.1(a)).

Next, b_1 and b_2 are compared to all the records in I'. Again, no matches are found, hence they are also moved to I'. Then, b_3 is compared against the records of I'. With a_1 it forms a match. Their records are merged and added to I, since this new record needs to be compared against all other records also (Figure 6.1(b) and Figure 6.1(c)).

In the next iteration, b_4 is compared against a_1 and a_2 without result, but against a_4 it forms a match. The merged record is also added to I (Figure 6.1(c)).

After several iterations we have the situation as depicted in Figure 6.1(d). The record $\langle a_1, b_3 \rangle$ does not match with a record in I' and hence is moved, but $\langle a_4, b_4 \rangle$ does match with c_2. The records are merged and the newly created $\langle \langle a_4, b_4 \rangle, c_2 \rangle$ is added to I.

Then, after another few iterations the last two records in I are compared to those in I'. No match is found and hence they also move to I' (Figure 6.1(e)). Then, I is empty and the algorithm terminates.

We like to reuse this algorithm. Hence, we will explore if we can satisfy the conditions in the next section.

6.2 Model

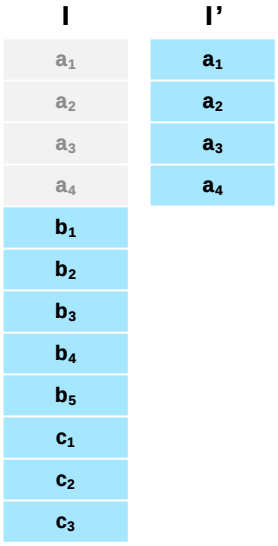
To be able to handle multiple sources in our model according to the R-Swoosh algorithm, we need to (re)consider some issues, followed by the adjustments we have to make in order to map R-Swoosh to our model.

6.2.1 Merge Result

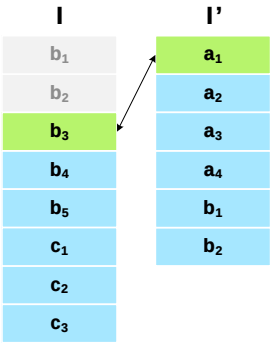
In our model so far, we only concentrated on the matching process. But by introducing multiple sources, we need to integrate profiles, because we need to compare merged records against other (merged) records. We can only make sensible design decisions about this if we know how a merged profile looks like.

There are several options for merging the profiles. For each attribute you can decide to take the union of the two source attributes, choose one of them or perform a more complex merge. Figure 6.2 illustrates this for the case where we merge a record with name {John Smith} and another with {Mr. J. Smith}.

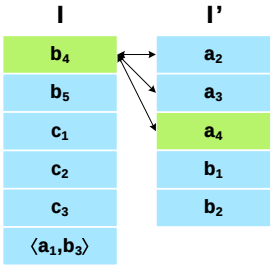
In this case, choosing for one of the names means you would lose the complete first name or an indication of the gender of this person. Performing a more complex merge that combines the information from both names would be ideal but not always as straightforward as in this example. It depends on the semantics of a certain attribute and hence this function is different for each attribute. To keep things as generic as possible we will use the union of the two attributes. This approach is suitable for all kinds of attributes.



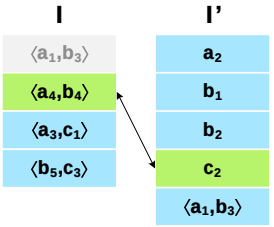
(a) Records a_1, a_2, a_3 and a_4 are each compared against the records in I' (without result) and then moved to I'.



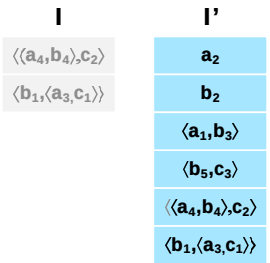
(b) Records b_1 and b_2 are also compared against the records in I' without result and are hence moved. Record b_3 matches with a_1.



(c) $\langle a_1, b_3 \rangle$ is added to I. Record b_4 is now compared and matches with a_4.



(d) After a few iterations we start comparing merged records against the records in I'. $\langle a_1, b_3 \rangle$ matches nothing and is moved, but $\langle a_4, b_4 \rangle$ matches c_2.



(e) In the last two iterations, the final records at I cannot be matched to any record in I' anymore and are moved to I'.

Figure 6.1: Some steps from the R-Swoosh algorithm.

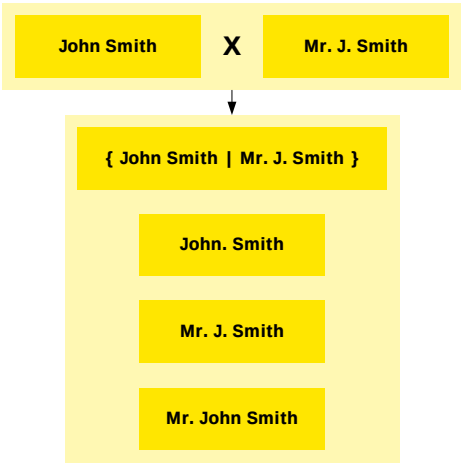


Figure 6.2: Different possibilities to merge names.

Taking the union of both records is not only the most generic solution, but also comes with other benefits. First of all and probably most important is that we satisfy the condition that our merge function is in the Union Class. With such a merge function we need to adapt our match function to be in the Union Class as well. We will discuss this in Section 6.2.3.

Second, we are able to control the similarity scores better. We will explain this with an example. Suppose we have a record p_1 with key value pairs {Name={John Smith}, Education={University of Twente, Delft University}} and a record p_2 with key value pairs {Name={Mr. J. Smith}, Education={University of Twente, Highschool X}}. In Figure 6.3 we see two different merge results. $\langle p_1, p_2 \rangle$ is the variant with union of values and for $\langle p_1, p_2 \rangle'$ a more complex merge function is used.

⟨p1,p2⟩

Name	John Smith	Mr. J. Smith
Education	University of Twente	University of Twente
	Delft University	Highschool X

⟨p1,p2⟩'

Name	Mr. John Smith
Education	University of Twente
	Delft University
	Highschool X

p3

Name	Johnny Smith
Education	University of Twente
	Highschool Y
	Highschool Z

Figure 6.3: Results of the matching function are dependent on the result of the merge function.

We now would like to compare p_3 against the merged record. Suppose the pairwise similarity score for two records is defined as

$d_{pw}(p_i, p_j) = \frac{1}{2} \cdot (x + \frac{|p_i.Education \cap p_j.Education|}{\mathsf{min}(\ |p_i.Education|, |p_j.Education|\ )})$, where x is a score for similarity in names. We will abstract from x in this example. For the 'Education' part of d_{pw} of the similarity score results can be different for each variant of the merged record. Let's compare p_3 against $\langle p_1, p_2 \rangle'$. Both records share one school and both have listed three schools, hence this gives $\frac{1}{3}$. If we would have compared p_3 to p_1 or p_2 alone, this would give $\frac{1}{2}$, which is a much higher score. Combining records can thus decrease similarity scores. We think that an average of the scores of separate comparisons is a much better representation of the similarity of the merged record. Comparing p_3 against the $\langle p_1, p_2 \rangle$ would then result in $\frac{1}{2} \cdot (\frac{1}{2} + \frac{1}{2}) = \frac{1}{2}$. This approach requires that values of the original records stay grouped in a merged record.

Definition 6.2 *A profile p_i consists of only one profile: the base profile. We say $b(p_i) = p_i$. A merged profile $p_k = \langle p_i, p_j \rangle$ consists of more than one base profile: $b(p_k) = \{p_i, p_j\}$.*

6.2.2 Confidence Scores vs. Match or Non-Match

A confidence score is attached to a record (or profile in our case) and can have different semantics in different domains. It could express to which extent the content can be trusted to be a representation of the real world, or it could represent the probability that this possible world is true. In [25] the authors took the notion of confidence scores into account. Although they abstracted from a function that would assign confidence scores to records, these scores influenced the ER process. We will give an example of this.

Suppose we have a record p_1 with confidence 0.8 and a record p_2 with confidence 0.7 and $p_1 \approx p_2$ is true. A merge function will merge these records into $p_3 = \langle p_1, p_2 \rangle$ with confidence 0.6. Apparently the merged record has a lower confidence score then the source records. This could be an indication of loss of quality or loss of information. Therefore, it could be wise not to delete the source records.

The similarity scores we used in our model are a form of confidence scores. Suppose we have two profiles p_1 and p_2. Based on the string matcher $\mathsf{asm}(p_i.attr, p_2.attr) = 0.88$ we decide that $p_1 \approx p_2$. Then, the similarity score 0.88 could be used as a confidence score for the merged record $\langle p_1, p_2 \rangle$. However, until now, this score was only necessary to decide if $p_1 \approx p_2$ holds. Once this decision is taken, the score is forgotten.

Using confidence scores in a multiple sources scenario, could lead to more complex situations. We will illustrate this with another example. Suppose we have three profile sites $X = \{A\}$, $Y = \{B\}$ and $Z = \{C\}$. We start the matching process for sites X and Y. $\mathsf{asm}(A.attr, B.attr) = 0.71$ and we decide $A \approx B$. If we then compare $\langle A, B \rangle$ with C from site Z, the similarity score is $\mathsf{asm}(\langle A, B \rangle.attr, C.attr) = 0.3$, which is not enough to be a match. However, $\mathsf{asm}(A.attr, C.attr) = 0.8$. It seems that $A \approx B$ and $A \approx C$, but not $\langle A, B \rangle \approx C$, from which you could conclude that either $A \approx B$ or $A \approx C$ is true, but not both. In this case $\langle A, C \rangle$ has a higher confidence score than $\langle A, B \rangle$. Should you then withdraw the previous decision?

To be able to choose the best match, you should delay the decision to the end of the process, after having compared all profiles from all sites against each other. In theory this would be the best option. However, this approach requires

huge memory and takes a very long time. Hence, we decided that we will not propagate confidence scores to a next step. Once a decision is made about a match, we stick with this decision. The profiles will be merged and the source profiles will be excluded from further comparisons.

6.2.3 Redefined Match Function

Our match function so far consists of determining the pairwise and network similarity scores for two profiles. For two complete sources, depending on the scores for the different candidate matches the matches were determined. This will not change except that we need to determine the similarity scores based on a merged profile. Below, we present the adapted similarity scores d'_{pw} and d'_{nw}.

Definition 6.3 *The pairwise similarity score d'_{pw} that can handle merged profiles is defined as follows:*

$$d'_{pw}(p_i, p_j) = \frac{1}{|b(p_i)| + |b(p_j)|} \cdot \sum_{b_i \in b(p_i)} \sum_{b_j \in b(p_j)} d_{pw}(b_i, b_j) \tag{6.1}$$

Definition 6.4 *The network similarity score d'_{nw} that can handle merged profiles is similar to the pairwise similarity score d'_{pw} and presented below:*

$$d'_{nw}(p_i, p_j) = \frac{1}{|b(p_i)| + |b(p_j)|} \cdot \sum_{b_i \in b(p_i)} \sum_{b_j \in b(p_j)} d_{nw}(b_i, b_j) \tag{6.2}$$

6.2.4 Reuse of R-Swoosh

From the previous sections, we already know that we satisfy the conditions to use the R-Swoosh algorithm. Unfortunately, we still have to do some adjustments. The R-Swoosh algorithm bases its decisions about matches on records from all sources, whereas we base ours on the records belonging to different sources. We think this approach results in a better matching process and hence we would like to adjust R-Swoosh to be compatible with this. The result in shown in Listing 6.2 and Listing 6.3.

In this algorithm we do not compare all records from all sources with all records from all other sources, but compare all records from one source with all records from all other records. We benefit from the fact that there are no duplicates within one source, hence we can skip several unnecessary comparisons.

Roughly speaking, it comes down to this: we take a source from S. If no sources are yet present in S', we move the complete source to S'. Then, we take the next source from S and for all sources in S', we compare all profiles. The comparison is listed in Listing 6.3. Only if we find a match, we remove the original (or base) profiles from the sources. For each combination of sources we create a new source to which the matched and merged profiles of these sources go. This source will be added to S when all profiles are compared. If we take a source from S that is a created source with merged profiles, the unmatchableSources() function will return all sources from which the merged profiles originate. This prevents the algorithm from doing unnecessary comparisons.

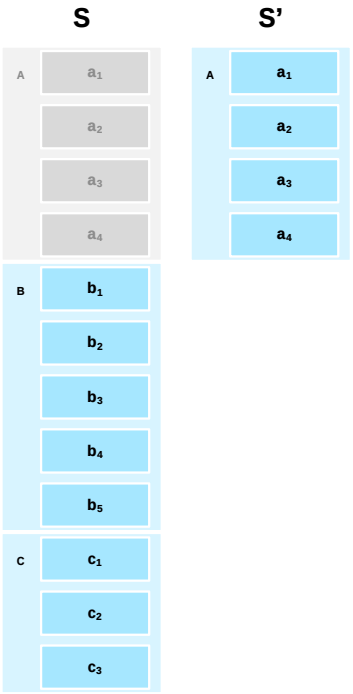
In Figure 6.4 we see the equivalent of the example for R-Swoosh, this time for the adapted R-Swoosh algorithm.

Listing 6.2: The adjusted R-Swoosh algorithm.

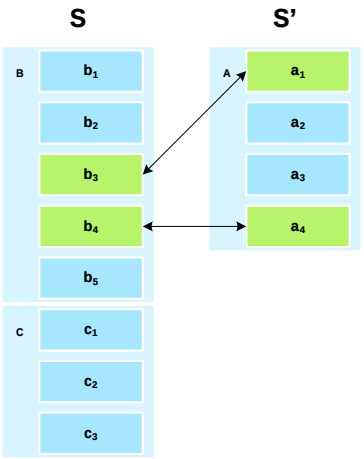
```
Input: a set S of sources
Output: a set S' of sources, S' = ER(S)
S' ← ∅
while S ≠ ∅
    currentSource ← a source from S
    remove currentSource from S
    while S' ≠ ∅
        otherSource ← source from S' − unmatchableSources(currentSource)
        newSource ← compareSources(currentSource, otherSource)
        if newSource ≠ ∅
            add newSource to S
        endif
    endwhile
    add currentSource to S'
endwhile
return S'
```

Listing 6.3: Comparing sources.

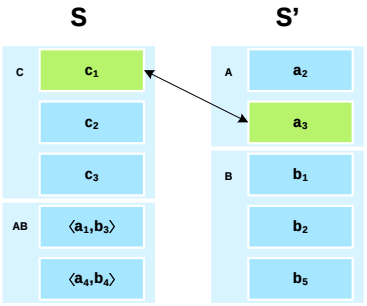
```
Input: a set firstSource of profiles, a set secondSource of profiles
Output: a set mergedProfiles of profiles
matches ← doMatchingProcess(firstSource,secondSource)
mergedProfiles ← merge(matches)
remove base(mergedProfiles) from firstSource
remove base(mergedProfiles) from secondSource
return mergedProfiles
```



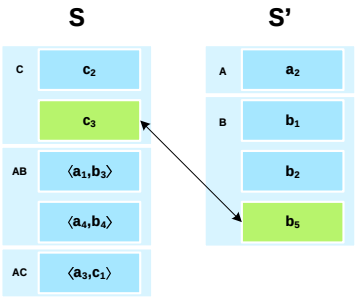
(a) S' was empty, hence site A was moved to S' immediately.



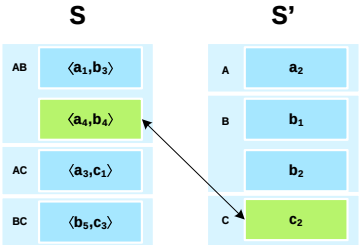
(b) Site B is compared against site A. Two matches are found. These will from a new source, which will be added to S.



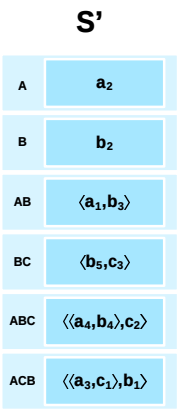
(c) C is then compared against the sites in S'. First A.



(d) Then, C is compared against B.



(e) The newly created source AB is also compared against the sources in S'.



(f) The resulting set S' after the algorithm is terminated.

Figure 6.4: Some steps of the adapted R-Swoosh algorithm.

6.3 Discussion

Since we succeeded in adjusting the R-Swoosh algorithm such that it can handle complete sources, we are able to implement this extension of the model as a layer on top of the model presented in the previous sections. We think this is a huge benefit.

Due to this model, the order in which we compare the different sources has become less important. Take a look at the example presented in Figure 6.5. The three profiles all belong to the same person. If we start comparing profile site A with profiles site C first, it is likely that (p_i, p_k) is not marked as a match. If we than compare A with B, and $p_i \approx p_j$ is concluded, the merged profile $\langle p_i, p_j \rangle$ still needs to be compared against site C, which results in $\langle p_i, p_j \rangle \approx p_k$. Since new merged profiles need to be compared again against all other sources, we found the desired result. Unfortunately, there are also cases in which the order is important, as we saw in the example presented in Section 6.2.2. However, although the impact of wrong decisions may be bigger than missed matches, we believe that this situation will not occur very often when the thresholds are high enough, since we expect that higher thresholds will filter out the most unlikely candidate matches immediately.

Since the order in which the sources are compared is not so important, it is quite easy to add a new source. This set of profiles can simply be added to the set of sources that are not yet compared against all other sources. This property of the multiple-source layer is a huge benefit. It makes the matching algorithm very flexible.

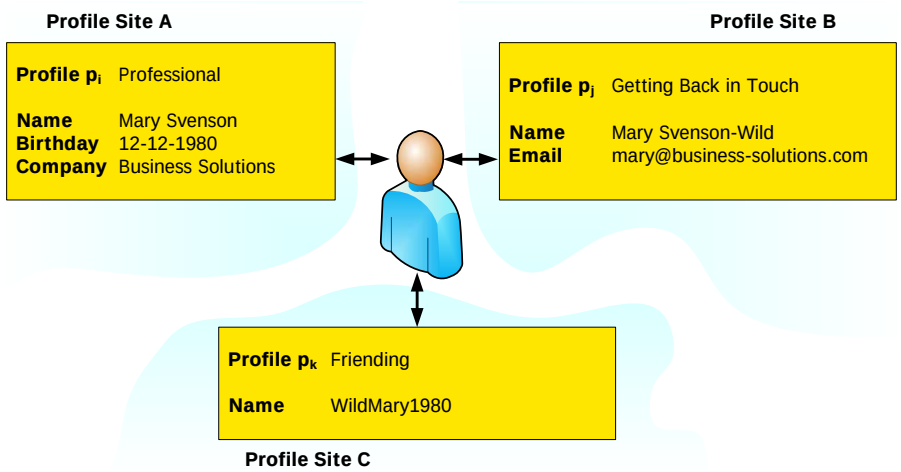


Figure 6.5: Three different profiles belonging to the same person.

Very interesting question is what will happen in case of a wrong match. What if two profiles p_i and p_j are no true match, but are marked to be a match by the model. We expect that it depends on what profile matches next to the merged profile. There are three possibilities: 1) $p_k \approx \langle p_i, p_j \rangle$, because $p_k \approx p_i$ is in fact a true match, 2) $p_k \approx \langle p_i, p_j \rangle$, because $p_k \approx p_j$ is in fact a true match or 3) $p_k \approx \langle p_i, p_j \rangle$ because of the combination of p_i and p_j, but neither (p_k, p_i) nor (p_k, p_j) is a true match. In the first two cases we expect that more correct

matches will match with the merged record, since this merged record contains for 67% correct information. The other 33% does not belong to the same person, but at least shows some similarities, otherwise it would not match in the first place. The third case is the worst case. The merged record contains information of three different persons and it is likely that if it matches another time with some other profile, that this profile belongs to yet another person. At this point we do not have any idea of how we can prevent these kinds of situations to happen.

The issue discussed above brings us to the next issue: how can we measure the correctness of our model? For just two sources this is easy. They either match or do not match. But now, we can have occurrences where two of the base profiles of merged profile are a true match, but the third is no true match with any of the other profiles. Is this match for 67% correct or is it completely incorrect? And what if we miss a match? For two sources, this is easy too. A true match is either found or missed. But in case of multiple sources, part of the total true match can be found or missed. How should this be expressed? This issue still needs to be addressed, when this layer on top of the model is implemented and tested. The measurements in case of two sources are presented in Section 7.2 in more detail.

As we can see, this extension of the model comes with some very interesting problems that need further investigation. Unfortunately, we were not able to include this part of the model into the prototype, hence we cannot draw any conclusions about the multiple source extension, with respect to the quality of the results.

Chapter 7

Experiments

In this chapter, we perform experiments with the prototype to see how our model performs. We vary the different parameters present in the model to come up with the best configuration and to be able to determine in what situations the model can be used.

Therefore, we elaborate on the parameters first. Then, we explain the measurements we use to compare the quality of the model in Section 7.2. After that, the results are presented in Section 7.3. We conclude with the best values for the different parameters in Section 7.4.

7.1 Parameters

The model contains quite a few input parameters that can be varied.

Pairwise Threshold (τ_{pw}) This is the minimum the d_{pw} score should be, to become a candidate match. It should have a value in the range $[0, 1]$.

Network Threshold (τ_{nw}) This is the minimum the d_{nw} score should be, to stay a candidate match. It should have a value in the range $[0, 1]$.

Pairwise Weight (ω_{pw}) The weight assigned to d_{pw} in the total similarity function d. The pairwise weight is a natural number > 0: $\omega_{pw} \in \mathbb{N}^+$.

Network Weight (ω_{nw}) The weight assigned to d_{nw} in the total similarity function d. The network weight is a natural number > 0: $\omega_{nw} \in \mathbb{N}^+$.

Method There are two ways of specifying which attributes are used for the matching process: natural or normal. These methods are explained in Section 7.1.1.

String Comparer One can choose among various string comparers for the comparison of attributes of profiles. The choices are:

- Block Distance (Token-based)
- Cosine Similarity (Token-based)
- Dice Similarity (Token-based)

- Euclidean Distance (Token-based)
- Jaccard (Token-based)
- Jaro (Character-based)
- JaroWinkler (Character-based)
- MongeElkan (Character-based)
- NeedlemanWunch (Character-based)
- Q-Grams Distance (Hybrid, Q = 3)
- SmithWaterman (Character-based)

Types (Optional) Specifies if the model should involve the network types.

Compensation (Optional) Specifies if the model should compensate for the fact that the profiles that are level-2 have an incomplete network. Without compensation, comparison with networks may give unexpected results. Therefore, you can run the experiments with compensation for these incomplete networks. The model will, at the point where the Precision, Recall and F-Measure are determined, take only the matches that are level-1 into account.

7.1.1 Natural vs. Normal

With method Normal you can specify on what attribute the profiles should be compared, and with which string matcher. Next to that, you can specify which string matcher to use. We mentioned the different string matchers already above. With respect to the attributes, the following choices are available:

- Name
- Email
- Birthday
- Address
- Schools
- Companies

On the other hand, there is the method Natural. This method mimics natural human behavior, i.e. when a human is asked to decide whether two profiles are similar, you can expect him or her to look at certain attributes, that uniquely identify a person, such as Name. First, the attributes Name and Birthday provide us with quite unique information about the owner of the profile (if provided). The attribute Email is expected to be reliable whenever it is provided by the user, but unfortunately this attribute is not unique per member. A person can have several email addresses and provide different ones for each profile page he or she owns. Finally, if both contain one or even more similar schools or companies, this provides us with even more evidence that these profiles belong to the same person.

In order to mimic this behavior, the method natural compares the names first. This yields a similarity score. Then, for all the other attributes just

mentioned, we also compare them. Eventually we do not take the weighted average of these scores, since the attributes that are not always provided can have a negative effect on the weighted average. Instead, we use the similarity score for the names as a base and add bonus points for having at least a certain similarity score for each other attribute. In this way, the other attributes only provide extra support. We experimentally determine which attributes should be bonus attributes.

7.2 Measurements

In order to quantify the results, we need proper measurements. We use measurements from the information retrieval field: Precision, Recall and F-Measure [34]. Figure 7.1 shows schematically what Precision and Recall represent. In this figure, the bounding rectangle represents D, the set of all pairs of profiles. In it there are two ellipses. The leftmost one represents all pairs that are marked as a match by the matching algorithm: D^{match}. The rightmost ellipsis represents the real world matches or true matches D^{true}.

Definition 7.1 *All pairs that we know for sure are true matches are contained in the set D^{true}, $D^{true} \subseteq D$.*

The intersection $D^{match} \bigcap D^{true}$ is the set of matches found by the matching algorithm that are matches in the real world as well. The bigger this set, and the smaller the sets $D^{match} - D^{true}$ and $D^{true} - D^{match}$, the more accurately the matching process performs.

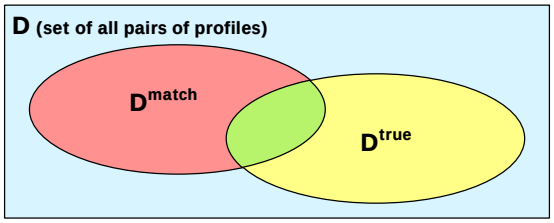


Figure 7.1: Graphical presentation of the measurements.

7.2.1 Precision

Precision measures the ratio between the true matches found and all found matches. In Figure 7.1 this is the green area, divided by the red and green area, or:

$$Precision = \frac{|D^{match} \bigcap D^{true}|}{|D^{match}|} \tag{7.1}$$

The Precision gives an indication of the number of false matches.

7.2.2 Recall

Recall measures the ratio between all found true matches and all true matches. In Figure 7.1 this is the green area, divided by the green and yellow area, or:

$$Recall = \frac{|D^{match} \bigcap D^{true}|}{|D^{true}|} \tag{7.2}$$

The Recall gives an indication of the number of missed matches.

7.2.3 F-Measure

The F-Measure is the weighted harmonic mean of the Precision and Recall. The traditional F-Measure is balanced and will have an equal weight for both Precision and Recall.

$$F = \frac{2 \cdot Precision \cdot Recall}{(Precision + Recall)} \tag{7.3}$$

There are variants that assign different weights to Precision and Recall. Depending on the purpose of the matching process it may be more important to have as few duplicates as possible in the integrated database (less missed matches, thus higher Recall) or as few wrongly integrated profiles as possible (less pairs marked as match wrongly, thus higher Precision).

7.3 Results

In Section 7.1 we presented the different parameters that can be varied in the prototype. For each parameter, we investigate its influencers on matching quality.

7.3.1 Pairwise Threshold and Network Threshold

The pairwise threshold τ_{pw} specifies how similar two profiles at least should be. The similarity is measured by a string matcher on certain attributes of the profiles. The higher the threshold, the likelier the two attributes of the profiles refer to the same entity. By increasing the threshold, the number of candidate matches will decrease, since it is more restrictive on the similarity of the attributes.

The network threshold τ_{nw} specifies how much overlap the networks of the two compared profiles should have. The overlap is the ratio of shared network profiles with respect to the smallest network of the two. By increasing the network threshold, a bigger overlap is needed for two profiles to be marked as a match.

For the experiments we varied the pairwise threshold from 0.50 to 1.00. We believe that a lower value for the pairwise similarity score is not desired, since the number of candidate matches increases enormously and hence the chance for wrongly marked matches.

The network threshold is varied from 0.00 to 0.60. A network threshold of 0.00 means that we do not put any restrictions on the value of the network similarity score, but we do perform the network comparison. In other words, we

do not eliminate candidate matches based on their network score, but for every candidate match, the network score is determined and added to the pairwise similarity score. Instead of on the pairwise score only, the total score is used to determine the matches, hence the results for $\tau_{nw} = 0.00$ can slightly differ from the variant in which we do pairwise comparison only.

For the experiments on the different thresholds, we choose the attribute Name to compare and Levenshtein as string matcher. We have chosen for this attribute, because it is a discriminative attribute, as we will see in Section 7.3.3 and Levenshtein also performs fine on all kind of attributes.

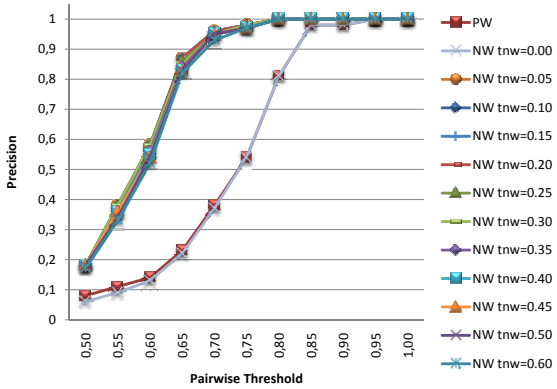


Figure 7.2: Precision *(Compared attribute: Name, String matcher: Levenshtein, Method: Normal, $\omega_{pw} = \omega_{nw}$).*

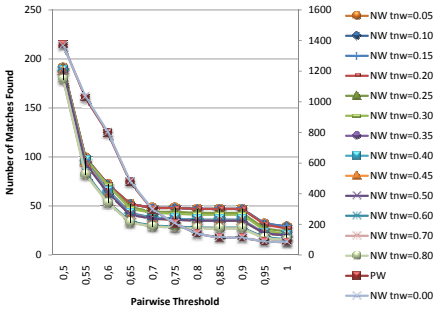
Figure 7.2 shows that with increasing pairwise threshold the Precision increases, as expected. The values for pairwise comparison only (PW) and for network comparison with $t_{nw} = 0.00$ (NW tnw = 0.0) are significantly lower than for those with a higher network threshold. This is probably caused by the additional network score, which has a much more positive effect on the total scores of candidate matches that are in fact true matches than other candidate matches. Remarkable though is that the value for the network threshold does not really seem to matter, since the Precision is almost the same for every $\tau_{nw} \geq 0.05$. But, with increasing network threshold, both the number of matches and correct matches decrease, as shown in Figure 7.3.

As we can see, the number of matches found for relatively high network thresholds is surprisingly high. It was expected that the number of matches found would decrease enormously with an increasing threshold τ_{nw}. Unfortunately, this is mainly caused by the high number of matches that involve level-2 profiles. We will elaborate on how these profiles negatively affect the results in Section 7.3.4.

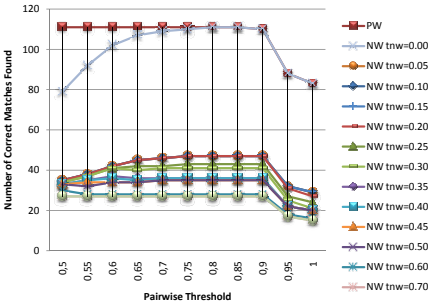
As we can see in Figure 7.3(b), the number of correct matches found for the process with network comparison is much lower than for pairwise comparison only. This is also shown in Figure 7.4.

The lower Recall values for the network comparison are due to the more restrictive conditions that need to be satisfied in order to be marked as a match.

As expected, the matching process with networks but without restrictions ($\tau_{nw} = 0.00$), behaves almost like the matching process without networks at all. Only we expected that it would perform slightly better. We believed that,



(a) The number of matches found for the different thresholds. Pairwise only (PW) and network comparison with $\tau_{nw} = 0.00$ (NW tnw=0.00) have a much larger number and are shown against the right vertical axis.



(b) The number of correct matches found. Note that the number of true matches is 111.

Figure 7.3: Number of (Correct) Matches Found *(Compared attribute: Name, String matcher: Levenshtein, Method: Normal, $\omega_{pw} = \omega_{nw}$).*

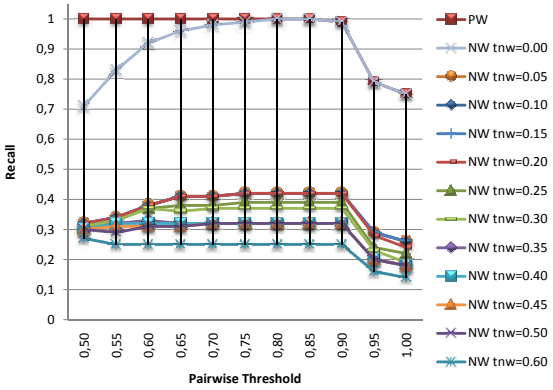


Figure 7.4: Recall *(Compared attribute: Name, String matcher: Levenshtein, Method: Normal, $\omega_{pw} = \omega_{nw}$).*

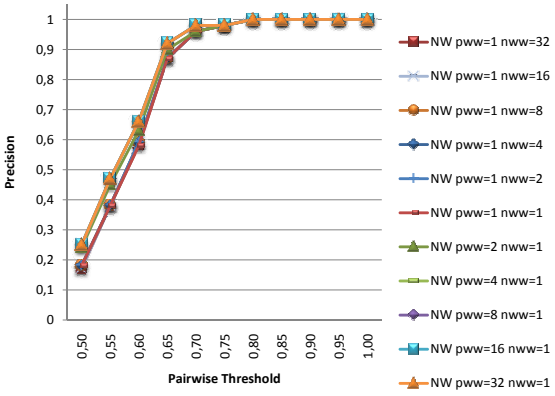
since it adds scores to the total scores and does not remove any of the candidate matches, it will have a positive effect on which matches are picked. Unfortunately, from Figure 7.3(b) we must conclude that it is in fact the other way around for lower values of the pairwise threshold. This can be explained though. For these lower values namely, candidate matches that are not true matches will also benefit from the network score. Sometimes this score will exceed the score of a true match, which results in wrong choices.

From the graphs we can conclude that in this setting the following thresholds are optimal: $\tau_{pw} = 0.90$ and $\tau_{nw} = 0.00$. But as we will see later on, we may expect better scores for the network similarity of matches, hence we will use $\tau_{nw} = 0.05$ in other experiments.

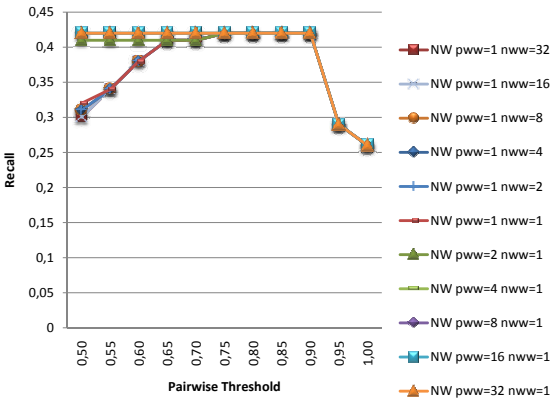
7.3.2 Weights

With ω_{pw} and ω_{nw} we can assign weights to the pairwise similarity score d_{pw} and the network similarity score d_{nw}. Since network scores in general are quite low, but tend to give better results on Precision, we expect that assigning a higher weight to the network score can improve the results.

We ran experiments to see what the effect of the weight on the results would be. In Figure 7.5 we can see that the effects are insignificant and even the other way around from what we expected.



(a) Precision with varying weights.



(b) Recall with varying weights.

Figure 7.5: Precision and Recall *(Compared attribute: Name, String matcher: Levenshtein, Method: Normal, $d_{nw} = 0.05$).*

Only for pairwise thresholds $\tau_{pw} \leq 0.70$ an increasing weight for the pairwise similarity score yields to a higher Precision and Recall. We ran the experiments again with $\tau_{nw} = 0.30$ and these results show the same pattern.

There is an explanation for this. Without the weight for the pairwise similarity score it could happen that a pair with not so high pairwise score, but very high network score was chosen above a pair with higher pairwise score and lower network score, while the latter is the actual true match. Let us illustrate this with an example. Suppose we have a candidate match (p_1, p_2) with

$d_{pw}(p_1, p_2) = 0.60$ and $d_{nw}(p_1, p_2) = 1.00$ and a candidate match (p_3, p_4) with $d_{pw}(p_3, p_4) = 1.00$ and $d_{nw}(p_3, p_4) = 0.40$. In the situation where $\omega_{pw} = \omega_{nw}$ is valid, pair (p_1, p_2) would be marked as a match, mainly due to the high network similarity score. But this high network score is in this case caused by p_1 having a network size of only one and this connected profile is similar to one of the connected profiles of p_2. The other pair that is in fact more similar (which follows from d_{pw}), has a much lower network similarity score, although $d_{nw} = 0.40$ is still a very high score for network similarity. By assigning more weight to the pairwise similarity, the "false" effect of the highest network score is made less important and now the correct candidate match is marked as a match.

The same phenomenon also appears when we perform the experiments with the option in which we compensate for incomplete networks. Although the weights do not behave as expected, this phenomenon confirms the assumption that pairwise similarity is more important than network similarity.

The higher pairwise weight is only helpful at lower values for the pairwise threshold, though. With lower pairwise thresholds, the set of candidate matches is much higher, which will slow down the algorithm. Moreover, we already decided that the optimal pairwise threshold is above $d_{pw} = 0.70$, for which a higher pairwise weight does not help. Hence, for the rest of the experiments we will keep using the configuration in which $\omega_{pw} = \omega_{nw}$.

7.3.3 Attributes and String Matchers

We can base the similarity scores of two profiles on the comparison of different attributes. We can choose among Name, Email, Birthday, Address, Schools and Companies.

To see how these attributes score on the true matches, we compared every attribute with every string matcher for each pair. In Table 7.1 we present the minimal and maximal value for the pairwise similarity score d_{pw} (and the distribution of the values are shown in Appendix D). A high minimal score means that for the comparison of this attribute the threshold can be at most as high as this score to include this true match in the list of candidate matches. Hence, we expect string matchers that yield to a higher minimal score for this attribute to perform better than other string matchers. Matches for which one of the attributes was not completed, where not included in these results.

Unfortunately, we cannot conclude which string matcher is most suitable for which attribute from this table only. If a string matcher assigns high scores to matches, there is a chance that true non-matches will be assigned a high score as well. Therefore, we need to take a look at how these strings perform on the complete data set.

We ran the experiments for all combinations of string matchers and attributes separately with $d_{pw} = 0.90$ and $d_{nw} = 0.05$. The results expressed in Precision and Recall are shown in Figure 7.6 for pairwise comparison only and in Figure 7.7 for the matching process with network comparison.

Name From Table 7.1 we may conclude that Jaro and JaroWinkler are the best choices, but Figure 7.6(a) shows that the Precision is much lower for these string matchers than for NeedlemanWunch and Levenshtein. This means that these string matchers assign high scores to true non-matches as well. Hence, Levenshtein and NeedlemanWunch seem to be the best choices for Name.

Table 7.1: The minimal and maximal pairwise similarity score per string matcher, per attribute for each true match.

	Name		**Email**		**Birthday**		**Address**		**Schools**		**Companies**	
String Matcher	min	max	min	max	min	max	min	max	min	max	min	max
BlockDistance	0,33	0,89	0	1	0	1	0	0,5	0,08	1	0	0,8
CosineSimilarity	0,35	0,89	0	1	0	1	0	0,58	0,09	1	0	0,82
DiceSimilarity	0,33	0,89	0	1	0	1	0	0,5	0,09	1	0	0,8
EuclideanDistance	0,18	0,67	0	1	0	1	0	0,29	0,04	1	0	0,55
Jaccard	0,2	0,8	0	1	0	1	0	0,33	0,05	1	0	0,67
Jaro	0,86	0,99	0	1	0,62	1	0,28	0,79	0	0,35	0,3	0,76
JaroWinkler	0,92	0,99	0	1	0,62	1	0	0,87	0,1	1	0,1	0,86
Levenshtein	0,8	0,96	0,1	1	0,38	1	0,04	0,38	0,09	1	0,03	0,65
MongeElkan	1	1	0,11	1	0,38	1	0,15	1	1	1	1	1
NeedlemanWunch	0,85	0,96	0,53	1	0,67	1	0,5	0,63	0,5	1	0,5	0,78
QGramsDistance	0,7	0,94	0	1	0,1	1	0	0,47	0,02	1	0,02	0,68
SmithWaterman	0,72	0,98	0,06	1	0,38	1	0,12	1	0,08	1	0,12	1

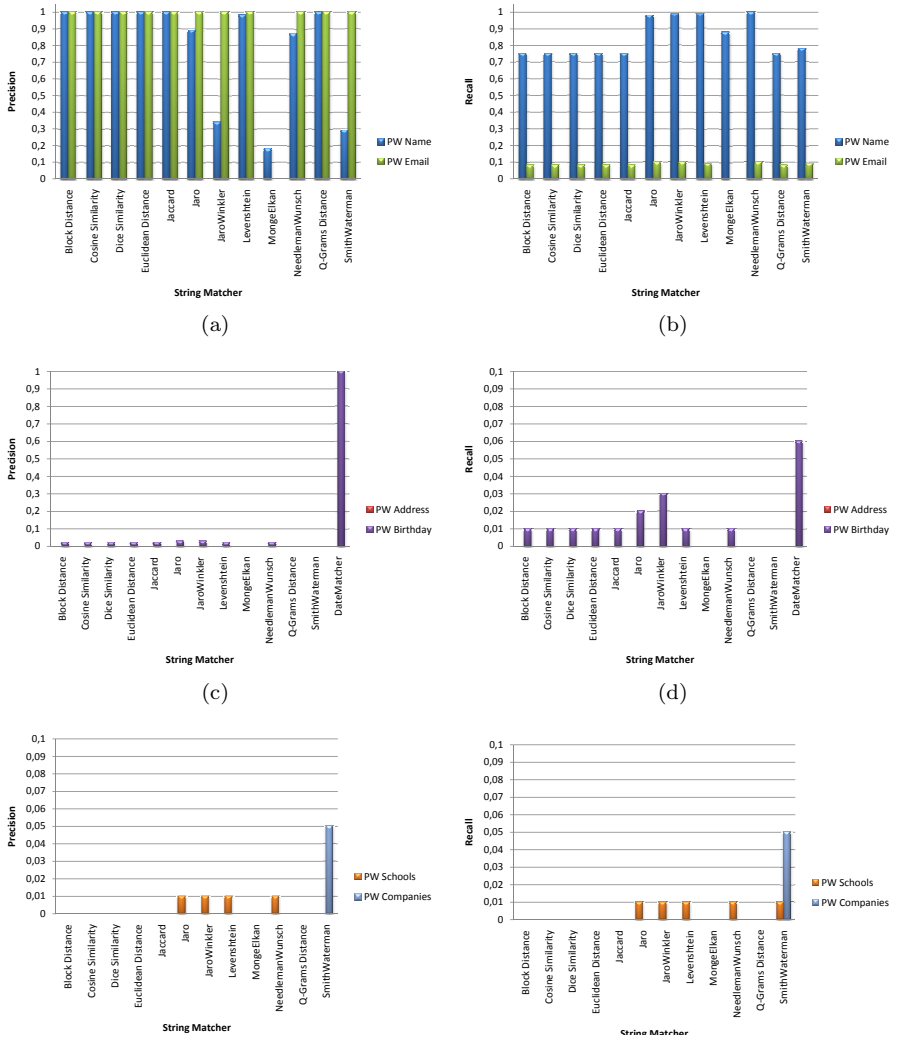


Figure 7.6: Precision and Recall for different attributes and for different string matchers, pairwise comparison only *(Method: Normal,* $d_{pw} = 0.90$ *and* $d_{nw} = 0.05$*,* $\omega_{pw} = \omega_{nw}$*).*

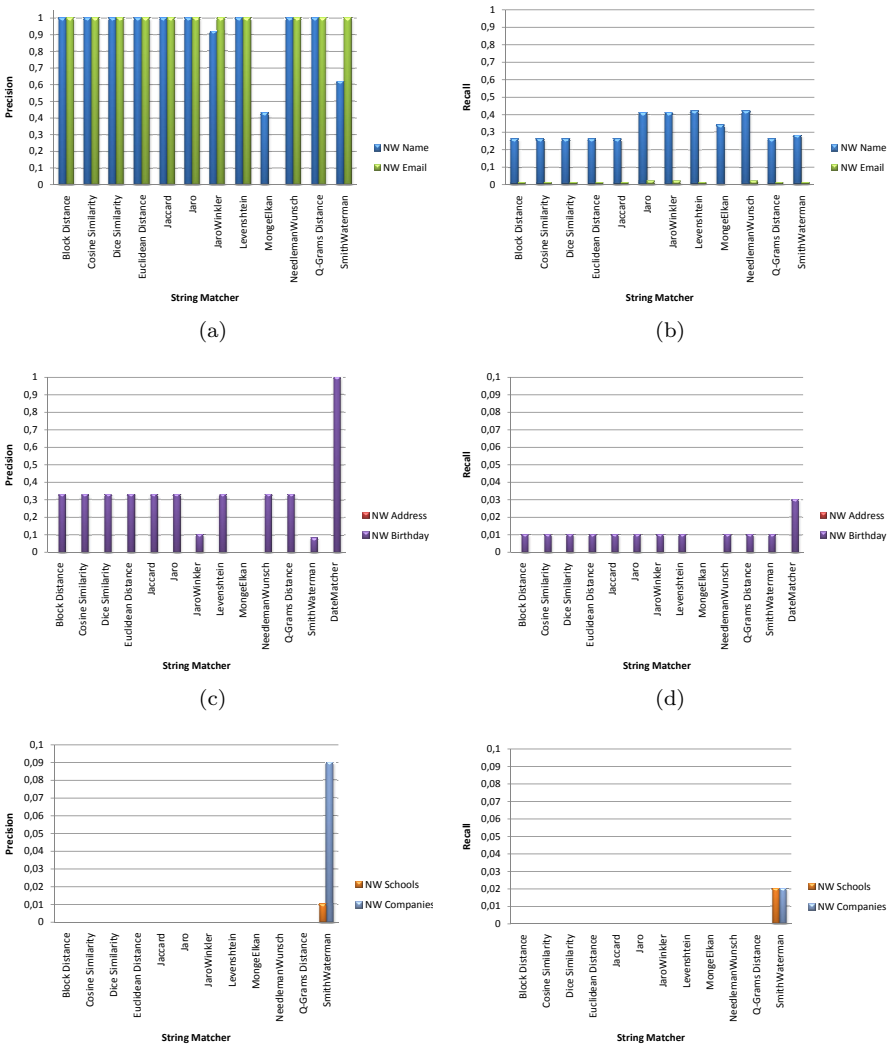


Figure 7.7: Precision and Recall for different attributes and for different string matchers, with network comparison *(Method: Normal,* $d_{pw} = 0.90$ *and* $d_{nw} = 0.05$, $\omega_{pw} = \omega_{nw}$*).*

We can also see that the Precision for the attribute Name is very good with and without network comparison. Unfortunately, the Recall decreases for this attribute with network comparison.

Email The experiments on the complete data set show that almost every string matcher results in similar results. From Table 7.1 we can see that for NeedlemanWunch every true match has a score of at least 0.53, hence this string matcher is very suitable for this attribute.

The Precision for the attribute Email is high with and without network comparison. The Recall decreases enormously for this attribute with network comparison. Many matches are missed because the email addresses are not so often visible for everybody, especially in case of Hyves profiles.

Address We performed the same experiments for Address, but in the experiments on the complete data set this attribute seems useless. This can be explained. In Hyves the field Address is your complete address or just the city you live in. A member is free to provide whatever he likes. At LinkedIn, you provide your zip code and LinkedIn shows the region to which it belongs on your profile. This is quite different from the address data at Hyves and the results confirm this. For a much lower value of the pairwise threshold, there are some candidate matches, but the pairwise similarity scores of these matches are the same for almost every candidate matching pair. With equal scores, sorting is useless and the matching process cannot pick the right matches.

From the tests on the true matches only it is confirmed that comparing the addresses will only result in a few distinct values, upon which our algorithm cannot base a good decision.

Birthday In Table 7.1 we see the highest minimal scores for Birthday with the string matchers Jaro, JaroWinkler and NeedlemanWunch. From the experiments on the complete set, it shows that the latter scores worse on the Recall. Hence, it seems that Jaro and JaroWinkler are the best suitable string matchers for this attribute.

Encouraged by the good results on the attribute Birthday already, we wanted to improve the results by handling a birthday not as a string, but as a date. Until now, a date like "1948-06-02" was very similar to "1998-06-03": only two characters are wrong. But the difference in semantics is very high, though. To avoid this situation, we introduced our own string matcher for dates, which will assign scores to similar days, months and years in a date. The results for this date matcher are shown in Figure 7.6(c), Figure 7.6(d), Figure 7.7(c) and Figure 7.7(d). For pairwise comparison only and the matching process with networks involved, the results are much better, mainly for the Precision. Unfortunately, the Recall stays very low. But if we look at the statistics for this data set, we see that for only 9% of the matches both profiles have a birthday completed. Thus, the maximum for Recall in these experiments for Birthday is only 0.09.

Schools and Companies Schools and Companies are attributes that certainly do not uniquely identify a profile, hence we expected to see fragmented results

for the true matches and low scores on the experiments with the whole data set. The results confirm these expectations.

Schools and Companies only show some results for certain string matchers, although the results are very low. Remarkably, they perform best with the SmithWaterman string matcher, which is a character-based string matcher.

If we perform the same experiments for the whole data set with lower pairwise thresholds, i.e. $d_{pw} = 0.70$, $d_{pw} = 0.50$ (see Appendix D) we see a shift towards the token-based string matchers with respect to the favorable string matchers. From Table 7.1 we expected to see some good results on the experiments with the NeedlemanWunch string matcher, but the experiments took far too much time to complete in a reasonable time.

7.3.3.1 Exclusion of String Matchers

MongeElkan is the string matcher that seemed very promising at first, looking at Table 7.1. However, it is suspicious that the scores for this matcher on attributes like Schools and Companies are perfect even if the attributes themselves do not match. Besides, MongeElkan performed very slowly, most of the times. Although we do not focus on performance of the algorithm, at the end it matters and hence we decided to brake off the experiments with MongeElkan when they lasted over an hour (some experiments with this string matcher lasted over 12 hours, while most experiments take less than 5 minutes).

Next to this string matcher, there were others that performed very slowly on experiments with low pairwise thresholds. These include NeedlemanWunch, Jaro and JaroWinkler.

7.3.3.2 Best Choices

From the results of the different attributes we can conclude that Name is the most discriminative attribute. Email and Birthday come next, but are not completed very often, especially birthdays at LinkedIn are completed rarely. Schools and Companies are not suitable as a primary attribute to match upon, but might increase Precision when used in combination with another more discriminative attribute. We applied this idea in the method Natural, explained next.

We performed these experiments again with $d_{pw} = 0.70$ and $d_{pw} = 0.50$. These results support our conclusion above. For the curious reader, we refer to Figure D.7 till Figure D.10 in Appendix D.

From these results is seems that for lower values of the pairwise threshold the Precision decreases whereas the Recall stays the same. Exceptions are the attributes Email and Birthday. For Email the Precision stays optimal for the token-based string matchers. For Birthday the Precision and Recall even increase for $\tau_{pw} = 0.50$ with the token-based string matchers. Also, our own specialized date matcher does not outperform the other string matcher any more.

The attributes School and Company show no real pattern and seem a bit unpredictable. However, both seem to benefit from the token-based string matchers with a lower pairwise threshold.

7.3.4 Compensation for Incomplete Networks

In Section 7.3.1 we encountered a strange problem. While we expected the number of matches found to decrease enormously when increasing the network threshold τ_{nw}, it did not. This effect is caused by the high number of matches that involve level-2 profiles. We must involve these profiles in our search for matches, since they form the network of the level-1 profiles. However, a candidate match containing two level-2 profiles, both having only one connection that happens to be a match as well, will yield a network score d_{nw} of 1. Of course, it is true that the complete networks of these profiles overlap in this case, but they influence the results in an undesired way. The network scores for the pairs of these profiles do not represent the situation as it would be with a complete data set. Hence, we have an option that can compensate for this. It will perform the matching process as it would do normally, but removes all matches that were marked by the process that do not consist of two level-1 profiles.

What you expect with this option is that the results are better and more accurate. Side effect of this option, however, is that the set of true matches is quite small, which makes it harder to conclude from the results.

In Figure 7.8 we see the results with the same configuration as for the experimental results in Figure 7.2 and Figure 7.4.

As we can see, both Precision and Recall have improved. For $\tau_{pw} \geq 0.70$ the Precision is maximal for pairwise comparison as well as for network comparison, independent of the network threshold. However, if we take a look at the Recall we see that for $\tau_{nw} \geq 0.25$ the number of missed matches increases. For values $\tau_{nw} < 0.25$ and $0.70 \geq \tau_{pw} \geq 0.90$ the Recall is 1, which is good.

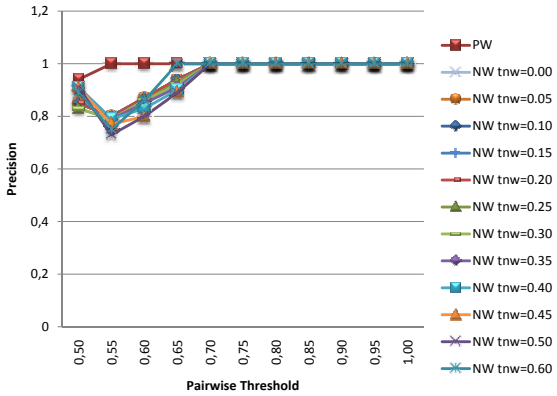
If we compare the F-Measure for both configurations, see Figure 7.9, we see that the range for which the F-Measure is 1 is much bigger with compensation then without compensation. Without compensation it is $\tau_{nw} = 0.00$ and $0.85 \geq \tau_{pw} \geq 0.90$ and with compensation $\tau_{nw} \leq 0.20$ and $0.70 \geq \tau_{pw} \geq 0.90$. It was to be expected for executing the matching process with this option that we can be more restrictive on the network similarity score without deterioration of the Recall.

7.3.5 Network Types

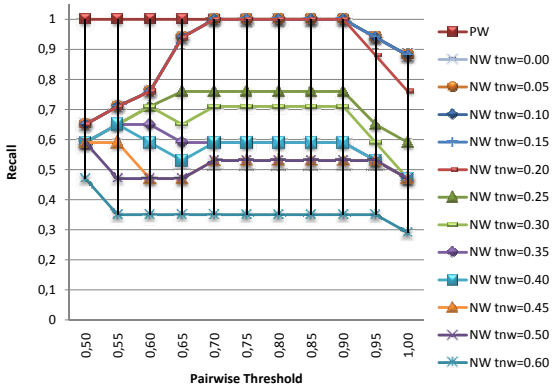
To see if we could extract even more evidence for a match from the network someone has, we introduced network types in Chapter 5. For each connection, we determined what the type of the connection is and we adjusted the matching process to handle these types.

In Figure 7.10 we show the results for the experiments. Again, we used the same configuration as we did for the experiments, for which the results were shown in Figure 7.2 and Figure 7.4.

We can see from the diagram that the Precision for the matching process with network types comparison gives better results than for pairwise comparison only. This is true for $\tau_{nw} \leq 0.40$. For $\tau_{nw} > 0.45$ Precision is 0. The Precision for the model with the network types extension is also better than the Precision for the model without this extension. Remarkable is that the Precision drops quite suddenly to zero and at such a high value for the network threshold. This is caused by the number of matches found. For each higher value for the network threshold, this number decreases enormously, but the ones found stay correct.

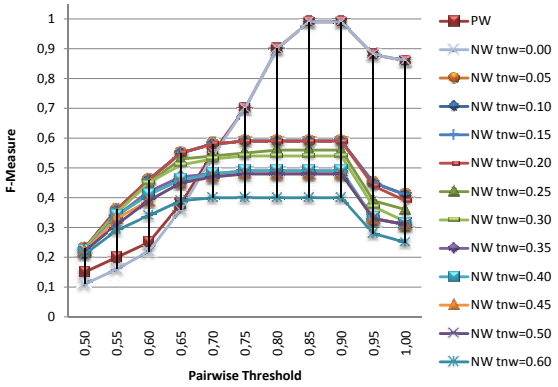


(a) Precision with compensation for incomplete networks.

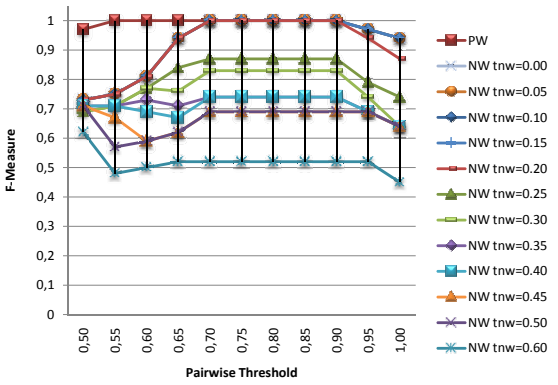


(b) Recall with compensation for incomplete networks.

Figure 7.8: Precision and Recall *(Compared attribute: Name, String matcher: Levenshtein, Method: Normal, $\omega_{pw} = \omega_{nw}$).*

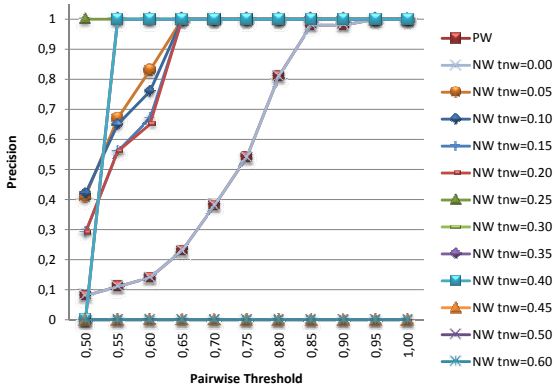


(a) F-Measure without compensation for incomplete networks.

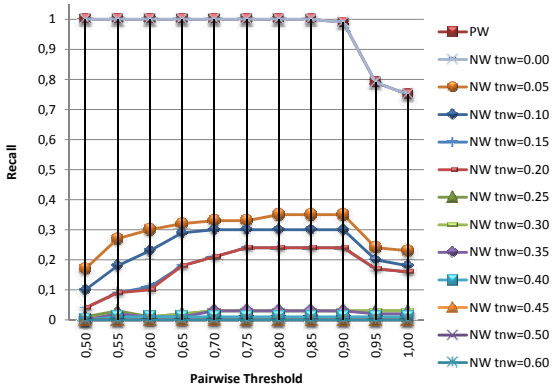


(b) F-Measure with compensation for incomplete networks.

Figure 7.9: F-Measure *(Compared attribute: Name, String matcher: Levenshtein, Method: Normal, $\omega_{pw} = \omega_{nw}$).*



(a) Precision for experiments with network types.



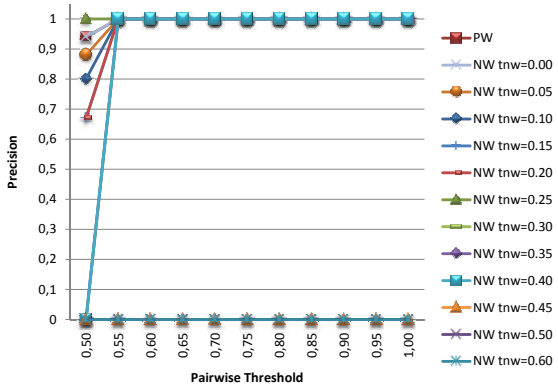
(b) Recall for experiments with network types.

Figure 7.10: Precision and Recall *(Compared attribute: Name, String matcher: Levenshtein, Method: Normal, $\omega_{pw} = \omega_{nw}$).*

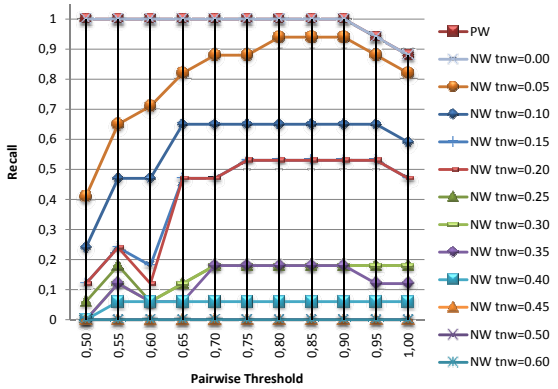
Recall shows us that with each higher value for the network threshold we miss more matches, indeed.

The Recall does not give those promising results for this model, unfortunately. The values for the Recall are at best (with $\tau_{nw} = 0.05$) almost similar to the normal model. For higher values of the network threshold, the Recall drops faster than for the normal model. This is caused by the even more restrictive conditions that have to be fulfilled before a candidate match will be marked as a match.

We can also perform this variant of the model with compensation for incomplete networks (with the same configuration). The results are are shown in Figure 7.11.



(a) Precision for experiments with network types.



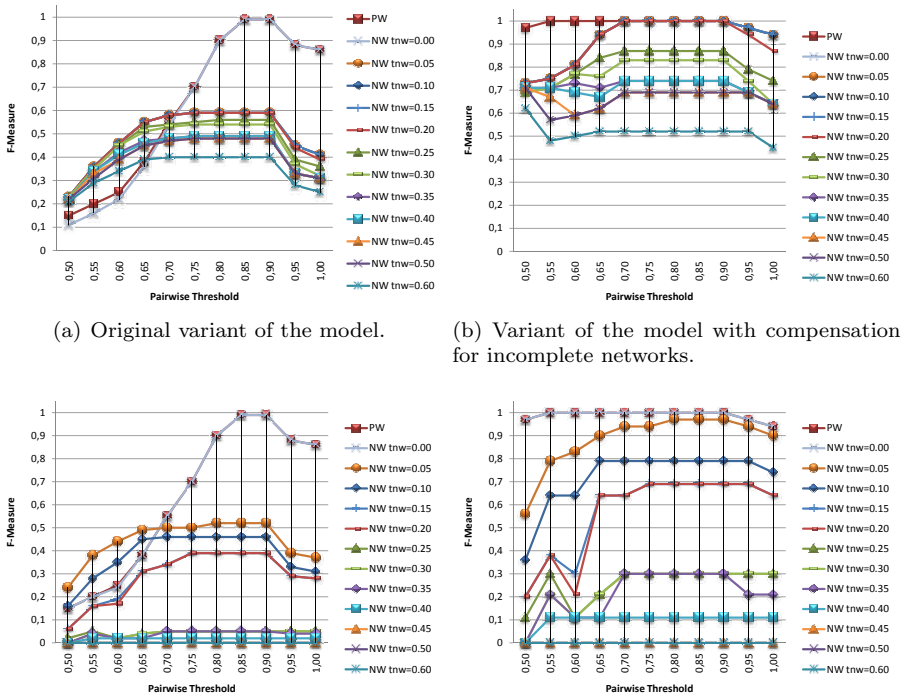
(b) Recall for experiments with network types.

Figure 7.11: Precision and Recall *(Compared attribute: Name, String matcher: Levenshtein, Method: Normal, $\omega_{pw} = \omega_{nw}$), Compensation.*

Notice that this time the values for the Recall have increased enormously in comparison with the network types variant without the compensation. This is no surprise.

For ease of comparing in Figure 7.12 we show the F-Measure values for the normal model (Figure 7.12(a)), with compensation (Figure 7.12(b)), with network types (Figure 7.12(c)) and with network types and compensation (Fig-

ure 7.12(d)).



(a) Original variant of the model.

(b) Variant of the model with compensation for incomplete networks.

(c) Variant of the model with network types.

(d) Variant of the model with both network types and compensation for incomplete networks.

Figure 7.12: F-Measures for different variants of the model *(Method: Normal, Attribute: Name, String Matcher: Levenshtein, $\omega_{pw} = \omega_{nw}$).*

Notice that the models with compensation perform better than the variants without compensation. The variants with network types perform only slightly worse than the variants without network types.

Finally, we should place a remark with these results. The matching process with network types is based on the types we have assigned to connections ourselves. We already pointed out that these types might not be accurate, which could have a negative effect on the results.

On the other hand, when you decide to connect to another person on a profile site, you do not want to be bothered with questions about the type of connection you are about to create. If a network type is required during the creation of such a connection, we believe that this answer will not always be so accurate either, or contain multiple types. Hence, we think that it could be worth it to always perform the task of assigning a type to each connection ourselves, not only in cases where no types are assigned by the user. That is, in case it will be helpful at all.

7.3.6 Method

So far, we performed all experiments with method Normal. In Section 7.1.1 we explained the differences between the methods Normal and Natural. In short, Normal compares a specified attribute with a specified string matcher for each pair of profiles. The method Natural compares for each two profiles the Name and adds a bonus to the similarity score for each time a comparison of the attributes Email, Birthday, School or Company yields to a good similarity score.

From the experiments in Section 7.3.3 and the distributions of the similarity scores in Appendix D we get a good hint of which string matchers to choose and which similarity scores will add a bonus to the score returned by the comparison of the names. Then, we need to find out how high a bonus should be to get optimal results. For the Name attribute the best string matchers are NeedlemanWunch and Levenshtein. The first is quite slow and will not always yield to results in a reasonable time. Email can be best matched with NeedlemanWunch or SmithWaterman or Levenshtein as alternatives. Birthday can be matched best with our DateMatcher or a token-based string matcher, such as BlockDistance. Schools and Companies can be matched best with NeedlemanWunch or SmithWaterman or CosineSimilarity as alternatives.

Matching Email and Birthday provides us with a very strong hint that profiles are matching. Because of this property, in contrast with Schools and Companies, the threshold in order to get a bonus for these attributes may be higher and the bonus as well.

Since this method introduces a lot of new parameters, we cannot test all values. However, we believe that, based on other experiments, we can estimate the attributes, string matchers and their thresholds quite accurate. For the attribute Name we chose NeedlemanWunch. From the experiments we know that if we compare more than one attribute with this string matcher, the experiments take too much time. Hence, we chose an alternative for all other attributes. For Email we chose QGramsDistance with the threshold for the bonus at 0.9, for Birthday the DateMatcher string matcher with the threshold at 0.9. For Schools and Companies we chose CosineSimilarity, both with the threshold at 0.4.

For the bonuses we did some experiments and the following configuration gives us the optimal results: compare Name with NeedlemanWunch, Email with QGramsDistance (threshold for bonus: 0.9, bonus: 0.25), Birthday with DateMatcher (threshold for bonus: 0.9, bonus: 0.25), Schools and Companies with CosineSimilarity (threshold for bonus: 0.4, bonus: 0.1).

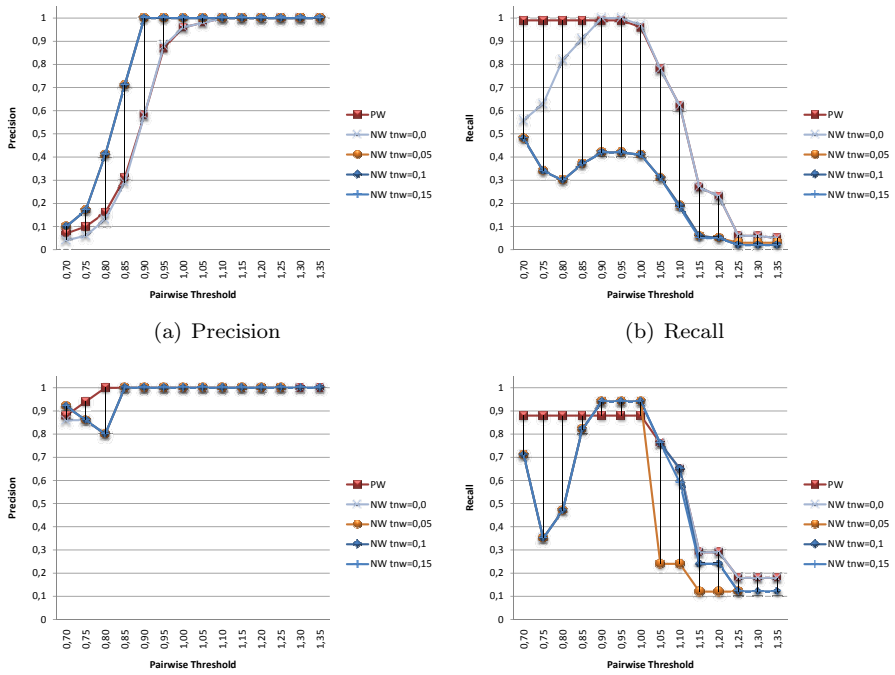
The results of the experiments with this configuration are shown in Figure 7.13. We expanded the range of the pairwise threshold, since the pairwise similarity score d_{pw} can exceed 1.

As we can see, the graphs show almost the same patterns for the Precision and Recall for the experiments with method Normal, presented in Section 7.3.1. Only a small shift towards a higher pairwise threshold can be detected. This is logical, since we add bonus scores to the similarity scores, thus same candidate matches are gained for higher pairwise thresholds.

For the experiments without the compensation, there are no major differences in results. Again, pairwise comparison only and network comparison with $\tau_{nw} = 0.00$ yields to better results than network comparison with $\tau_{nw} > 0.00$. For the experiments with compensation for incomplete networks, small changes are detected. The Recall never reaches the value 1 anymore, but the experiments

with network comparison score better on Recall than pairwise comparison only.

This configuration scores slightly worse than the method Normal. But in cases where the people do not complete their names so properly, this configuration could become more valuable, since it involves other attributes as well. The best values for the pairwise and network thresholds are: $0.90 \geq \tau_{pw} \geq 1.00$ and $\tau_{nw} = 0.00$.



(a) Precision

(b) Recall

(c) Precision with compensation for incomplete networks.

(d) Recall with compensation for incomplete networks.

Figure 7.13: Natural *Names with NeedlemanWunch, Email with QGramsDistance (threshold 0.9, bonus 0.25), Birthdays with DateMatcher (threshold 0.9, bonus 0.25), Schools and Companies with CosineSimilarity (threshold 0.4, bonus 0.1).*

7.3.7 Beyond the Limitations of the Data Set

As we may conclude from the previous section, even in a very much tuned configuration our network model does not improve the simple pairwise model significantly. This is quite disappointing, as it seems intuitively that this network extension of the model can contribute to better results. We believe that this is caused by the limited data set.

One limitation of our data set is that it does not contains ambiguous profiles. In the real world it will happen very often that people will have the exact same full name. However, with retrieving data just two levels deep, this chance was very small. But this situation is an example case in which the social networks will distinguish the different entities.

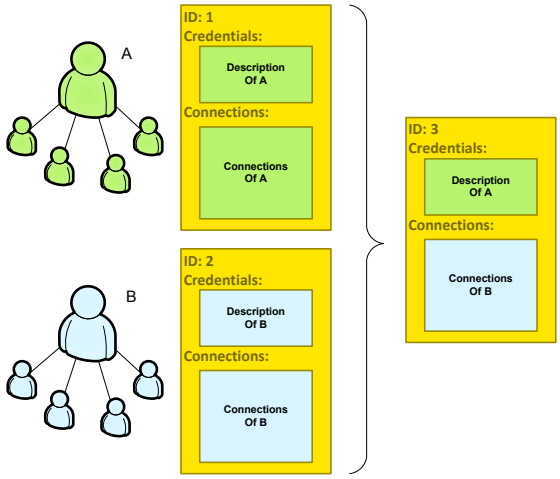


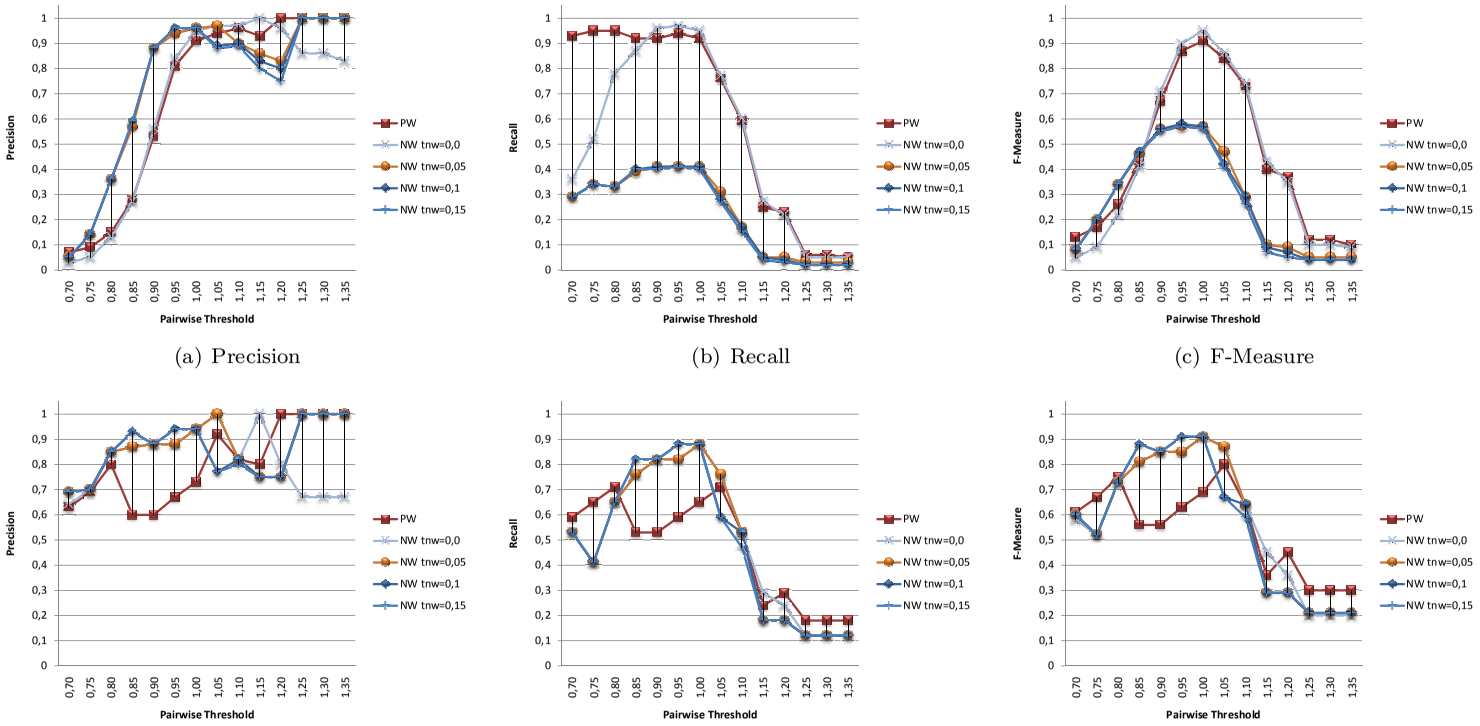
Figure 7.14: We created fake, ambiguous profiles containing a copy of the credentials of one profile and the connections of another profile.

We were unable to retrieve a data set that could prove this, hence we modified the first data set slightly. We created some new, ambiguous profiles by copying the credentials of one already existing level-1 profile and connecting it with all connections of another level-1 profile, illustrated in Figure 7.14. The profiles from which we used the credentials and the connections were chosen such that they did not belong to the same social network of the central profile, e.g. the level-0 profile. Each time we created a fake profile, one of these used profiles was in the network of colleagues and the other in the network of classmates. This avoids that the network of the fake profile shows quite some overlap with the original profile. We did this with 7 Hyves profiles and 6 LinkedIn profiles.

Now, with this new data set, we performed the experiments with the optimal configuration presented in Section 7.3.6 again. The results are shown in Figure 7.15.

For the experiments that were executed without compensation for incomplete networks, there is no significant change. However, the experiments with compensation show huge improvements, as expected. We see that for the pairwise threshold with values $0.85 \geq \tau_{pw} \geq 1.00$ our network model performs more than 20% better than the simple model.

The strange flaw in the graph of the simple model can be declared. Since the fake profiles share the exact same credentials as the original profiles, the fake and original profiles have the same similarity score d_{pw}. Since the decision for being a match is based on only the pairwise similarity score in the simple model, one of the two profiles will be chosen without any favor. Sometimes, this decision is correct and sometimes it is not. It is unpredictable which one will be chosen. This instability shows in the graph, both at Precision and Recall. The graphs of the network model are pretty stable and hence we can say that the network model is in this case significantly better.



(a) Precision

(b) Recall

(c) F-Measure

(d) Precision with compensation for incomplete networks.

(e) Recall with compensation for incomplete networks.

(f) F-Measure with compensation for incomplete networks.

Figure 7.15: Experiments performed with modified data set. This data set contains ambiguous profiles with non-similar networks to mimic a more natural situation. *Method: Natural; Names with NeedlemanWunch, Email with QGramsDistance (threshold 0.9, bonus 0.25), Birthdays with DateMatcher (threshold 0.9, bonus 0.25), Schools and Companies with CosineSimilarity (threshold 0.4, bonus 0.1).*

7.4 Concluding Remarks

We performed experiments on the prototype in order to see how our model with networking comparison performs with respect to the simple model. The values for the different parameters were varied to detect a good configuration.

First, we explored the effects on the results with varying pairwise and network similarity thresholds. With increasing pairwise similarity threshold, the value for the Precision also increases. The Recall is quite stable, but drops down with values for $\tau_{pw} > 0.9$. This is valid for both the simple model as for the network model, but the network model scores better on the Precision and the simple model on the Recall. This is because the network model is stricter on when a candidate match can be marked as a match.

In the network model it was also possible to assign weights to the pairwise and network similarity score. However, no significant improvements could be measured. Hence, we used equal weights in all other experiments.

Next to weights, we tested which string matcher we could use best and for what attribute. The attributes Name, Email and Birthday are the most discriminative, but only the first attribute is completed always. The best string matcher to use for Name and Email is NeedlemanWunch. Levenshtein is a good alternative. For Schools and Companies perform best with a token-based string matcher.

To overcome one of the limitations of the data set, we introduced the option to compensate for incomplete networks. Since the level-2 profiles only have a network of size 1, this return affect the results strangely. By excluding the matches that involve level-2 profiles, we get a more realistic picture of the matches found. The experimental results support this. Results for both Precision and Recall improved enormously for the network model. Still, they do not exceed the results for the simple model.

Another option we introduced was network types. We wondered if this could be of any help. Our expectations were not really high, but it seems that this option was not so bad at all, however it did not exceed the results of the network model without types.

All experiments mentioned above were performed with the method Normal. With this method, you can specify the values for each parameter. Next to this method, there is the method Natural. This is a more advanced configuration at which you can specify with which string matcher you want to compare the Names. But next to this attribute, Email, Birthday, Schools and Companies are compared as well, each with a fixed string matcher. For each extra attribute that returns a high score, a bonus is added to the pairwise similarity score. The best configuration for this method found is: Names with NeedlemanWunch, Email with QGramsDistance (with a bonus of 0.25 for scores over 0.9), Birthdays with DateMatcher (with a bonus of 0.25 for scores over 0.9) and Schools and Companies with CosineSimilarity (with bonus of 0.1 for scores over 0.4). The differences with respect to Normal are not significant, hence it might be a better to choose for Normal with comparison on the attribute Name since it only performs one attribute comparison per pair of profiles. In case the attribute Name is not completed as reliable as in this data set, the method Natural is expected to outperform Normal, since it relies on more than one attribute.

Unfortunately, the results so far were disappointing. We believed this was

caused by another limitation of the data set, the lack of ambiguous profiles. After we modified the data set to contain some of these ambiguous profiles, we performed the experiments again and the results show significant improvements on the results, indeed.

In general we can conclude that network comparison can improve the results for the matching process. However, in some cases, the simple matching model works fine already. The network model defines more conditions that need to be met in order to be a match, hence network comparison is more precise. With respect to the Recall, network comparison almost always scores worse on this than pairwise comparison only. This is because the addition of the network comparison eliminates candidate matches. This step can result in the elimination of correct matches. The network model can only perform better than the simple model on Recall if it eliminates no correct matches and yields to higher network similarity scores for correct matches, such that more correct matches are marked as match by the process, which was the case in the modified data set.

Chapter 8

Related Work

Entity Resolution (ER) is a popular issue and many researches address it. In our research we addressed a very specific case of ER, namely the matching of profiles with network support. In this section we will describe the related research and discuss how it relates to ours. We start with the theoretical research concerning ER and end with topics more related to this particular case. This chapter is not exhaustive, since there is just too much literature on ER, thus we only describe the approaches most similar to ours, and the ones which could help us improve our work further. The focus thereby lies on the effectiveness instead of efficiency of approaches, as in our own approach.

8.1 Entity Resolution

ER addresses the issue of deduplication. It uses smart ways to detect duplicates in data and to resolve them. In Chapter 3 we briefly discussed what causes duplicates to exist in data and what issues come with ER.

In [8], the authors divide different ER solutions into two different groups: 1) supervised and 2) unsupervised solutions. Supervised approaches need labeled training sets or predefined thresholds to base their decisions on. Unfortunately, the variety of patterns that can be observed from the real world can hardly be captured in a training set and therefore these solutions are quite limited. Unsupervised approaches avoid human intervention by using clustering algorithms that group together items with high similarity.

Actually, there is a third group of ER solutions, namely semi-supervised. A semi-supervised approach uses a small set of labeled data and a set of unlabeled data. The training phase is then skipped and together with the labeled data, the unlabeled data is resolved. We will not discuss this group of ER solutions in detail.

Our approach belongs to the group of supervised approaches, since we manually tuned the different parameters.

In Section 8.1.1 and Section 8.1.2 we will discuss supervised respectively unsupervised approaches. In Section 8.1.3 we will briefly describe different distance metrics.

8.1.1 Supervised Approaches

In this category of approaches the learning-based, distance-based and rule-based solutions belong. Felligi and Sunter [15] have laid important foundations in the field, by reducing the ER problem to the following simple problem: there are two classes: a class M with matches and a class U with non-matches. Each pair needs to be assigned to one of these classes.

The learning-based solutions use large training sets in which all pairs are labeled with "match" or "non-match". Each pair has a comparison vector that represents the comparable attributes of the two items in the pair. For the classes M and U, the assumption is that the density functions are different. Now, the problem of ER can be treated as a Bayesian inference problem. We will not discuss these solutions in detail.

The distance-based solutions do not need training data. In these approaches a distance metric is defined between data items. Based on the distance between two items, a decision is made about whether or not this pair is a match. For this decision, a threshold is needed. This threshold can be set by making a good estimation, but fine tuning this threshold can improve the results. Therefore, using some kind of training set might be desired.

Rule-based solutions use rules from domain knowledge to base decisions on. With respect to personal data, if two items contain the same Social Security Number (SSN) and birthday, you can conclude that the items refer to the same person.

Our approach is mainly distance-based, but uses rules as well. For instance, we have more confidence in corresponding names and emails than in corresponding schools and companies as an indication that two profiles refer to the same person.

In all approaches mentioned above, decisions are made for each pair in isolation, which is a more classical approach. In recent research the trend is to eliminate the naive assumption that these decisions are independent. We will discuss this next.

8.1.1.1 Collective Entity Resolution

In [31] the authors use a collective model for ER. Their idea is to propagate evidence. We will illustrate this with an example from their article, which is shown in Table 8.1. Let's take record 1 and 2. It is decided that these records are duplicates. Due to this decision we know that "KDD-2003" and "9th SIGKDD" refer to the same venue. As we propagate this knowledge we can decide that records 3 and 4 are duplicates as well, whereas this would not be clear without this knowledge. While the process is progressing, the knowledge grows. Information that was not available for the first records will be available at the end of the process. Hence, it is important that this process is performed iteratively.

The results with this form of ER are much more accurate, compared with the classical approaches. In our case, this approach could be helpful for the attributes Schools and Companies, since names of schools and companies often appear in many string representations (with abbreviations or additional suffixes). Unfortunately, it is not necessarily true that the value for these attributes are referring to the exactly same schools or companies for a true match, as opposed

to the example. Therefore, this approach could lead to incorrect knowledge, hence it is not suitable in our particular case.

Table 8.1: Records 1 and 2 are duplicates. These duplicates provide evidence for records 3 and 4 to be duplicates as well.

Record	Title	Author	Venue
1	Record Linkage using CRFs	Linda Stewart	KDD-2003
2	Record Linkage using CRFs	Linda Stewart	9th SIGKDD
3	Learning Boolean Formulas	Bill Johnson	KDD-2003
4	Learning of Boolean Expressions	William Johnson	9th SIGKDD

The authors of [4] perform collective ER, too. They present an approach whereby the database remains unresolved, but relevant records are resolved at query-time. In previous work they showed that collective ER improves accuracy, but mapping this approach to perform online is not straight-forward. Moreover, collective ER comes with quite some additional computational costs, which is not desired at query-time.

During query execution they use attribute expansion and hyper-edge expansion to extract the relevant records. The query returns some records and based on a certain attribute, other records containing the same value for this attribute are extracted as well. Hyper-edge expansion will extract all records that refer to the same data items the records refer to that are already extracted. This can be done iteratively. The set of extracted records will grow very fast, even at a small depth. They show experimentally that their approach improves the F-Measures significantly. Query-time consumption of the algorithm is the biggest issue that needs to be tackled. Their approach of performing ER at query time is very interesting however.

In our approach, we determine all scores independently from all other scores, or decisions already made. However, in the network model we base similarity on the overlap of the networks of two profiles. Moreover, in the phase where we determine which candidate matches are marked as a match, we remove candidate matches that are impossible due to decisions already made. Hence, our approach is a form of collective ER.

8.1.1.2 Manual Postprocessing

Another direction within supervised ER is one that requires manual postprocessing. Felligi and Sunter themselves [15] introduced a reject region, which is a class of pairs for which the decision cannot be made automatically. Human intervention is needed to resolve these hard cases.

In Figure 8.1 the idea of manual postprocessing is illustrated schematically. The picture comes from [9]. At the start, with a blocking or indexing technique, the pairs that are definitely not a match are filtered. The rest is compared against each other and assigned to one of the three classes. The possible matches need manual postprocessing and will then be assigned to either the class with the matches or the non-matches [10], [19].

The authors of [33] base their work on the same principle: they built their data integration on the rule of thumb that it takes about 90% of the time to solve the remaining 10% hard cases. They perform the integration but leave

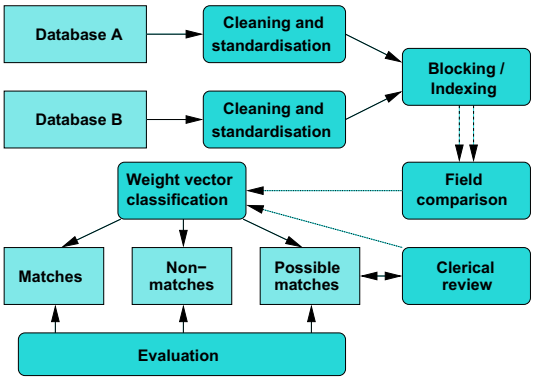


Figure 8.1: General schema of ER with clerical review (manual postprocessing) [9].

in the uncertainties they encounter during the integration. These uncertainties can be resolved by user-feedback at query time.

They perform so-called probabilistic data integration. When doubt arises about the integration of some data items, the different possibilities, each with a corresponding probability attached, will be stored as part of the integration process (hence the name). Whenever a user queries this uncertain data, multiple possibilities will be returned. If the user is sure that he or she knows the correct answer, he or she can give feedback. This user-feedback will then be used as evidence to resolve these items.

A remarkable difference between this matching process and ours it that uncertainty can appear at different levels of granularity. We take a look at items as a whole and decide if two items are the same, based on the similarity scores of one or more attributes. In the approach described in [33] they compare whole items against each other. If two items are probably the same, but doubt arises at a certain attribute then it is possible to capture only this uncertainty as different possible values for this attribute, instead of saving two possible values for the complete merged data item.

Important question is whether an approach with manual postprocessing could be helpful in our case. This depends on the purpose of the resulting integrated data. Minimal prerequisite for such a system is that there is at least a user. In cases where the resulting data will be further processed without human intervention, this method is not suitable. Moreover, for each user that processes or queries the data who is requested to give feedback on these data, the user should be capable of providing correct feedback, i.e. in our case, this person should know the entities in real life. This indicates that the data set should be small, or there should be many users (usually the latter).

8.1.2 Unsupervised Approaches

The idea behind unsupervised approaches is that similar comparison vectors correspond to the same class [14]. Clustering techniques use these vectors to group very similar data items together. In some approaches the formed clusters represent the entities while in other approaches the clusters only contain very similar items that still need processing.

Our model also uses clustering. We only use the clustering phase to divide the work to be done in smaller parts, i.e. if we would perform the determination of the matches on the complete set of candidate matches, the same matches would be returned. However, the resulting clusters do contain profiles that are very similar.

The authors of [8] tackle the problem of single link clustering approaches with global distance thresholds. In this approach, duplicates in a set are detected by clustering. Each record from this set is initially a cluster. For each two clusters, the distances are measured. Distances between clusters can be an aggregate function of the similarity scores in the cluster, e.g. average, minimum, maximum etc. The two clusters with the smallest distances are then grouped into one cluster. The distances need to be updated and then the two clusters with the next best (smallest) distance are grouped. The clustering terminates when no distance between two clusters satisfies a global defined threshold anymore. Ideally, when this threshold is chosen well, each cluster represents one entity. The problem with this approach is that it is hard to globally specify this threshold. They illustrate this with the example shown in Table 8.2. The first two records are duplicates and their distance will not be very high. The last two records will not score high on distance either, but these records are no duplicates. What a good threshold is depends on the duplicates and the other records that are very nearby but in fact no duplicates.

Table 8.2: Records in a data set. 1 and 2 are duplicates, 3 and 4 not.

ID	Artist	Title
1	The Beatles	A Little Help From My Friends
2	Beatles, The	With a little help from my friend
3	4th Elemynt	Ears/Eyes
4	4th Elemynt	Ears/Eyes - Part II

In their research the authors try to make the decision of what the value of this threshold should be more flexible. They build their ideas on the following two observations: 1) duplicated records are "closer" to each other than to others and 2) the local neighborhood of duplicate records is sparse. They developed an approach that can detect this variable threshold and did experiments with it. It shows that the accuracy improved enormously.

In [16] they use two variants of clustering that both lead to sets of clusters that need further processing. With the "by diameter" variant, the clusters are not disjoint and hence automatic processing is very complex and not advisable. With the "by transitive closure" variant, the clusters are disjoint. Here a cluster contains the transitive closure of the symmetric relationship that two data items have a similarity score that satisfies a certain threshold. This variant is very similar to our clustering approach, except that our clustering method contains the transitive closure of the symmetric relationship that two candidate pairs contain one equal profile.

In [33] they perform clustering in order to enhance scalability. Their clustering algorithm is similar to ours, except that they can handle clustering at different levels of granularity whereas we only perform clustering on profile-level. This is possible since they allow uncertainty at attribute-level, as already mentioned in Section 8.1.1.2. From their experimental results it seems that

clustering does indeed contribute to the scalability of the integration.

8.1.3 Distance Metrics

With the distance-based approaches, mentioned in Section 8.1.1, a distance metric determines how "close" two items are. The technique most often used is comparing corresponding attributes of the items based on their string representation. As mentioned in Section 3.1.1, there are several categories of these distance metrics: 1) character-based, 2) token-based and 3) hybrid. In fact there are more categories, including phonetic approaches. Soundex [29], [30] is the most well-known variant in this category. This method is based on equality of the sound of spoken words.

We will not discuss each distance metric in detail, as we already discussed the string matchers in Section 3.1.1. In this section we will describe some comparison studies performed on the string matchers. Unfortunately, these studies are quite rare.

In [12] the authors compared several string matchers with the help of their own implemented toolkit. Results showed that MongeElkan performed well among the character-based approaches. Jaro and JaroWinkler also perform well, but are faster than MongeElkan. TFIDF performed very well among the token-based approaches. They tested some hybrid approaches as well and the combination of TFIDF with JaroWinkler performed the best on average.

In [36] the author tested which string matcher would work best for a name matching task on the US census data. Here again, the JaroWinkler performs well. It should be noted that there is no general rule for the application of string matchers. This highly depends on the kind of data [5]. Our own research supports this. We've seen in the experiments that it depends on the attribute which string matcher you can use best.

8.2 Exploiting (Social) Network Relations

Apart from social networks, networks in general have been researched a lot. And since social networks are gaining so much popularity and even attention from the research field, the results from this effort are applied to social networks. In [17] a survey is presented of recent link mining research and future directions. They acknowledge that it is important to combine the work from different fields and present work that already follow this approach. They defined different categories in order to discuss the different link mining tasks. These tasks include "object ranking", "object classification" and also "object identification" or ER.

The tool D-Dupe, presented in [6], uses networks to perform ER. In the domain of scientific publications they used the co-authorship relation to detect duplicates. For each author in their data set they determine the network of co-authors with whom this author has written an article. In this network of co-authors, if two authors in the network have very similar names and the titles of the articles they wrote are very similar too, then these authors are probably the same person. Their information will then be merged. The network will be updated and can raise new possible matches. This approach differs in that they use one source in which duplicates are present. This means that you can check the network of one author for duplicates. Since the duplicates appear within one

network, similarities between co-authors (or their associated information, such as article title, journal etc.) provide a very certain hint that these co-authors are in fact the same person. Nonetheless, this tool leaves the user to decide whether or not two authors match, it only provides hints. For small sets this is feasible.

8.2.1 Aggregators

From the perspective of the profiles sites themselves, there are other initiatives to exploit the social network relations. Since there are many people that have profiles at multiple profile sites, there was a need for linking techniques. From the user perspective there was a need for so-called "social network aggregation", which provides a mechanism for maintaining multiple profiles from one interface. These aggregators are not concerned with the ER problem themselves, since the requests for integration of the profiles comes from the user. Examples of these aggregators are SocialThing![1], FriendFeed[2], Strands[3] and SecondBrain[4].

If a user thinks it is useful to let other people know who he relates to and which other profiles he might have, the *FoaF*[5] specification will be helpful. With FoaF (Friend-of-a-Friend) you can easily achieve this, by adding some meta-data to your home page for instance. This is best illustrated with an example from the site of Social Graph API[6]. In Figure 8.2 is a schematic version of the example.

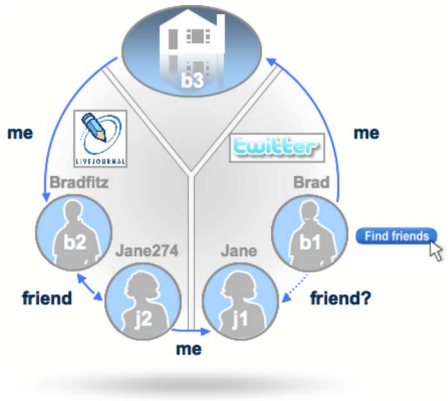


Figure 8.2: Illustrating example of FoaF

This example starts with Brad, on the right. He wants to know if there are people on Twitter he might know. Brad has a home page linking to his Twitter profile, but also to his LiveJournal profile, were he is called 'Bradfitz'. On LiveJournal he has a 'friend' link to 'Jane274'. This friend 'Jane274' has a link on LiveJournal pointing to her Twitter profile. So, clicking the 'find friends' button will result in 'Jane' as a candidate friend.

[1] `http://socialthing.com`, last visited 25 August 2009
[2] `http://friendfeed.com`, last visited 25 August 2009
[3] `http://www.strands.com`, last visited 25 August 2009
[4] `http://secondbrain.com`, last visited 25 August 2009
[5] `http://www.foaf-project.org/`, last visited 23 April 2009
[6] `http://code.google.com/intl/nl/apis/socialgraph/docs/`, last visited 22 april 2009

Although it seems to solve the ER problem in the case of profile pages, we do not expect it to be used very much, since it exists for several years, but not many large profile sites are supporting this. Besides, these links only work if someone is willing to cooperate. We do not expect that every member will do that.

Next to these initiatives, there are search engines that try to collect as much personal information as possible from these profile pages about a person. One of these sites is the Dutch "wie-o-wie". Unfortunately, they only match on names, which is very restrictive.

8.3 Concluding Remarks

ER is a much discussed and researched topic. There are many variants of approaches possible, but there are only a few basic ideas. These ideas include similarity measures and thresholds, clustering and knowledge rules. Our variant is a combination of several of these ideas, combined with some specific domain knowledge. Most important difference is that deduplication is not necessary within one source, but only among different sources. This means that you do not have to compare all profiles with all profiles which is a huge benefit.

The notion of clustering can differ from one research to another. In many researches a cluster is a set of items referring to one entity, i.e. clustering is the end of the matching process. In our case, clustering is an intermediate step to divide the following work in smaller pieces, for scalability reasons. From one cluster more than one match can be returned, which is different from the aforementioned approach. The idea of a variable threshold might be interesting in the future, especially since the variety in this "human data" is not predictable. Hence, choosing a good threshold is difficult.

If aggregators gain popularity, then it will be easier for people search engines to perform the matching. Then, there might still be a need for such matching techniques to integrate the results from a people search engine with customer data storage.

Chapter 9

Conclusions

There are many people who have a profile at more than one profile site. On these sites you can build a social network by connecting to other members of the site. If a person has multiple profiles, the social networks of these profiles are likely to show some overlap, i.e. there are profiles from both social networks that refer to the same person in real life. Based on this observation, we developed a model that uses this network overlap to determine if two profiles belong to the same person. Our motivation for this is that we thought that a model with social network support would improve the results of a model without network support. In order to test this we 1) retrieved a data set with real world data from the internet, 2) built both a simple model and a network model and 3) built a prototype to experiment with.

First, we needed a suitable data set. The data set was supposed to contain realistic personal data, including the usual data flaws that arise when people are anxious to complete all data, forget to complete some fields or when data is getting outdated. Unfortunately, there is no benchmark available that contains such data. Generating our own data set was not a good option since we believe it is not possible to generate realistic data flaws. Hence, we decided to crawl our own data set from two profile sites on the internet. Unfortunately, due to practical reasons it was not possible to crawl more than two levels deep from the starting profile. Moreover, due to this restriction on the crawling, there are not so many ambiguous profiles in the data set.

The base for the model is the comparison of one or several attributes of the profiles. Each profile from one source site was compared against each profile from the other source. This comparison results in a pairwise similarity score for each possible pair. Pairs with a score lower than a certain threshold are eliminated. The simple model will then determine which of these remaining (candidate) matches are matches, based on the similarity scores. The network model will compare the networks of candidate matches first, in order to determine which of them are matches. Comparing the social networks of the two profiles in a candidate match will yield a network similarity score. Based on both similarity scores the network model will decide which of them are matches. For scalability reasons we also added a clustering step.

The model was then tested with the prototype we built. By varying the different parameters we have gained a better understanding of the effect of the

network model on our data set. Mainly the network model is more restrictive on when to call a pair a match. Precision will in most cases benefit and Recall not at all, which is not really a surprise. In general, the simple model seemed to work quite well already and the question is whether it is worth the extra steps of the network model to get a higher Precision. The good news is that the amount of uncertainty was not as high as we expected, which is good for both models.

On the other hand, the data set has some limitations that negatively influence the results. By adding an option that compensates for the incomplete networks of the collected level-2 profiles in the data set, we retrieve more realistic results. With this option turned on, the results for the network model improved as expected. Besides, as already mentioned, the data set only contains no ambiguous profiles. We added some fake ambiguous profiles to mimic this situation in the data set. By performing the experiments on this modified data set, the results for the network model improved enormously. Moreover, and more important, the network model outperformed the simple model, because with this modified data set, the networks make the difference between a match and a non-match.

Although we had some problems with finding a realistic data set, we still have a very good idea of when to use the network model. We will elaborate on this in the next subsection.

9.1 Recommendations

Now we know how the network model behaves (with respect to the simple model) we will present below in which situation the network model can be suitable/preferable or not. The network model is:

- suitable in cases where high precision is required. In cases were decisions are made based on the integrated items, the number of mistakes should be kept as minimal as possible. We already mentioned the credit granting example. In this case, mistakes should not be made.

- suitable if the simple model results in many high scores. Network comparison could help reduce this number. Suppose we search for all people with a certain first name on several profile sites. If we want to perform the matching process on these sets of profiles, many ambiguous profiles will exist and the network model can make the difference.

- suitable in cases where networks serve another purpose, for instance social studies, marketing based on the (purchasing) behavior of a group. Some adjustments to the model might be necessary, depending on the exact purpose.

- not suitable in cases where high Recall is required. Take a police investigation for example. Suppose they want to list suspects (with alias detection), you do not want to discard options too fast. In this case, a large number of candidate matches is desired. Human intervention is required to make the final decisions.

Still, work is needed to be able to use the model in practice. In the next section we present some issues that need, or are interesting, for further exploration.

We did not pay attention to the performance in terms of speed. The algorithms need optimization, because the network comparison phase consumes much time. Pruning of cases in which (complete) network comparison is not necessary could be an option. Note that the use of the pairwise threshold is already an example of pruning, since it discards those pairs that are likely not to match.

9.2 Future Work

The network model already showed some promising results. However, there exist several interesting issues that still need exploration in the future to improve the network model or to integrate it in a more practical environment.

Improving the Network Model During the experiments, we discovered some strange effects that some deviant values can have on the outcome of an experiment. For instance, if we compare the networks of two profiles: one having a large network and the other having only one or a few connections. In Section 4.1.2 we reasoned that we could better divide the number of overlapping profiles by the size of the minimal network. However, strange situations still occur, see Section 7.3.1. Similar to the situation mentioned above are the effects that total scores have. Sometimes total scores are very high, due to a high pairwise or network similarity score, but not both. Sometimes, this leads to strange effects. We think that taking the mean of both similarity scores could correct this.

Optimization Next to the minor changes proposed above that could improve the accuracy of the results, the network model needs improvements that will improve the time-complexity and scalability. Performing network comparison is a very costly operation. Therefore, it would be very useful to think of a more efficient implementation. Moreover, pruning cases in which no (complete) network comparison need to be performed is an option. For instance, if there is a cluster for which the variation of the pairwise similarity scores is high, there is a great chance that picking the pairs with the highest scores are the correct matches.

Practical Setting In this research we were not able to test the network model in a more practical setting. In order to do so, adjustments need to be made to the model. First of all, before the network model can be used for an application the multi-source integration needs to be tested. Moreover, the model now works on batches of profiles, i.e. we have a large data set, without any filtering. In practice, such a data set is too large to operate on. Therefore, some filtering must take place on beforehand.

If you are interested in a target group, you want to retrieve a large data set that satisfies certain conditions. During the retrieval part, you can filter on these conditions in order to keep your data set "clean". Interesting question

is whether you retrieve the complete network of a person that satisfies this condition or only the connected profiles that satisfy these conditions as well.

If you are interested in only a small number of people, you would like to search for profiles that are likely to belong to them, without searching the complete internet. This requires some search function on the different profile sites so that you can search not only on name, but on other attributes as well. You want this search function to come up with more than one result, so you can compare the results with results from other sites as well and find your own result based on the comparison of these profiles (and their networks). The expansion methods proposed in [4] would be useful here. Unfortunately, only a few profile sites provide a suitable search function.

Collecting More Evidence Another issue that could be investigated is the collection of more evidence with more unconventional methods. You could think of tracing the surfing behavior of members. Someone who logs in at a certain profile site and next to another profile site is likely to have a profile at both sites. Of course, privacy concerns play an important role here.

Benchmark Independent from our own research, for future research in this direction the existence of a publicly available benchmark would be very useful. Unfortunately, due to privacy concerns they do not exist. However, we think that it is possible to create a data set with generated personal data. Maybe, this data set will not be useful for all purposes, but it will at least be very helpful in the start of a research. Moreover, people can expand this benchmark to serve their own purpose.

Bibliography

[1] E. Adar and C. Ré. Managing Uncertainty in Social Networks. *IEEE Data Engineering Bulletin*, 30, 2007.

[2] A. Barabasi. *Linked: the New Science of Networks*. Perseus Publishing, 2002.

[3] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. Whang, and J. Widom. Swoosh: a Generic Approach to Entity Resolution. *The VLDB Journal*, March 2008.

[4] I. Bhattacharya and L. Getoor. Online Collective Entity Resolution. In *The 22nd National Conference on Artificial Intelligence (NECTAR Track)*, volume 2, pages 1606–1609, 2007.

[5] M. Bilenko, R.J. Mooney, W.W. Cohen, P. Ravikumar, and S.E. Fienberg. Adaptive Name Matching in Information Integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.

[6] M. Bilgic, L. Licamele, L. Getoor, and B. Shneiderman. D-dupe: an Interactive Tool for Entity Resolution in Social Networks. In *IEEE Symposium on Visual Analytics Science and Technology*, pages 43–50, 2006.

[7] M. Chau, P. Lam, B. Shiu, J. Xu, and J. Cao. A Blog Mining Framework. *IT Professional*, 2009.

[8] S. Chaudhuri, V. Ganti, and R. Motwani. Robust Identification of Fuzzy Duplicates. In *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*, 2005.

[9] P. Christen. A Two-step Classification Approach to Unsupervised Record Linkage. In *Proceedings of the sixth Australasian conference on Data mining and analytics*, pages 111–119, 2007.

[10] P. Christen and K. Goiser. Quality and Complexity Measures for Data Linkage and Deduplication. *Quality Measures in Data Mining*, 43:127–151, 2007.

[11] W.W. Cohen. Integration of Heterogeneous Databases without Common Domains Using Queries Based on Textual Similarity. *SIGMOD Record*, 27(2):201–212, 1998.

[12] W.W. Cohen, P. Ravikumar, and S.E. Fienberg. A Comparison of String Distance Metrics for Name-matching Tasks. In *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWEB-03)*, pages 73–78, 2003.

[13] I.F. Cruz and H. Xiao. *Complex Systems in Knowledge-based Environments: Theory, Models and Applications*, volume 168, chapter 4: Ontology Driven Data Integration in Heterogeneous Networks, pages 75–98. Springer Berlin / Heidelberg, 2009.

[14] A.K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios. Duplicate Record Detection: a Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[15] I.P. Felligi and A.B. Sunter. A Theory for Record Linkage. *J. Am. Statistical Assoc.*, 64(328):1183–1210, 1969.

[16] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. An Extensible Framework for Data Cleaning. Technical Report RR-3742, INRIA, 1999.

[17] L. Getoor and D.P. Diehl. Link Mining: a survey. *SIGKDD Explorations*, 7(2):3–12, 2005.

[18] L. Gravano, P.G. Ipeirotis, H. Jagadish, N. Koudas, L. Muthukrishnan, S. Pietarinen, and D. Srivastava. Using Q-grams in a DBMS for Approximate String Processing. *IEEE Data Engineering Bulletin*, 24(4):28–34, 2001.

[19] L. Gu and R. Baxter. *Selected Papers from AusDM*, chapter Decision Models for Record Linkage, pages 146–160. Springer LNCS 3755, 2006.

[20] P. Jaccard. Distribution de la Flore Alpine dans le Bassin des Dranses et dans Quelques Régions Voisines. *Bulletin del la Socit Vaudoise des Sciences Naturelles*, 37:241–272, 1901.

[21] M.A. Jaro. Probabilistic Linkage of Large Public Health Data Files. *Statistics in Medecine*, 14:491–498, 1995.

[22] M. Leida. *Toward Semantics-aware Ontology Mediated*. PhD thesis, Università Degli Studi di Milano, 2008.

[23] M. Lenzerini. Data Integration: a Theoretical Perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems*, pages 233–246, New York, NY, USA, 2002. ACM.

[24] V.I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[25] D. Menestrina, O. Benjelloun, and H. Garcia-Molina. Generic Entity Resolution with Data Confidences. Technical Report 2005-35, Stanford InfoLab, 2005.

[26] P. Mika. *Social Networks and the Semantic Web, Computing for Human Experience*. Springer Science+Business Media, LLC, 2007.

[27] S.B. Needleman and C.D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *J. Molecular Biology*, 48(3):443–453, 1970.

[28] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society. *Nature*, 435:814–818, June 2005.

[29] R.C. Russell. R.C. Russell Index, US patent 1,261,167, 1918.

[30] R.C. Russell. R.C. Russell Index, US patent 1,435,663, 1922.

[31] P. Singla and P. Domingos. Multi-relational Record Linkage. In *ACM SIGKDD Workshop on Multi-Relational Data Mining*, 2004.

[32] Author Unknown. Comparison of the XML model and the Relational Model. From the DB2 Documentation of IBM, `http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.xml.doc/doc/c0023811.html`, last visited: 14 September 2009.

[33] M. van Keulen and A. de Keijzer. Qualitative effects of knowledge rules and user feedback in probabilistic data integration. *The VLDB Journal*, Special Issue Paper:1–27, 2009.

[34] M. Weis, F. Naumann, and F. Brosy. A Duplicate Detection Benchmark for XML (and Relational) Data. In *SIGMOD Workshop on Information Quality for Information Systems (IQIS)*, 2006.

[35] W.E. Winkler. Overview of Record Linkage and Current Research Directions. Technical report, US Bureau of the Census Research, 2006.

[36] W.E. Yancey. Evaluating String Comparator Performance for Record Linkage. Technical report, Statistical Research Report Series RRS2005/05, US Bureau of the Census, 2005.

[37] Z. Ziegler and K.R. Dittrich. Three Decades of Data Integration - All Problems Solved? In *18th IFIP World Computer Congress (WCC 2004): Building the Information Society*, volume 12, pages 3–12, 2004.

Appendix A

Original Data Sources

On the next pages we present the labels of the data fields as you can find them when you complete a profile at the profile sites LinkedIn, Hyves, Facebook and Live Spaces.

We kept the names of the labels as close as possible to the original names. Some information can be completed with more than one data item (such as schools), which is presented by the extra columns. For instance, with LinkedIn, Hyves and Facebook you can enter on which schools and/or universities you followed a study, i.e. more than one if necessary. At the Live Space profile, you can only enter for one study what you did.

LinkedIn	
Data	**Consists of**
given-name	
familiy-name	
location	
e-mail	
phone	
phone-type	
address	
bd-day	
bd-month	
bd-year	
marital status	
experience	job
	company
	current
education	school-name
	finished
websites	website-name
	website-url
connections	connection-name
group	group-name

Hyves	
Data	**Consists of**
voornaam	
achternaam	
e-mail	
woonplaats	
adres	
huisnummer	
postcode	
mobiele telefoonnummer	
geslacht	
geboorte-dag	
geboorte-maand	
geboortje-jaar	
relatie	
school	school-naam
MBO/HBO/WO	instellings-naam
bedrijven	bedrijfsnaam
website	
vrienden	vriend-naam

Figure A.1: Labels of the data fields that belong to LinkedIn and Hyves profiles.

Facebook	
Data	**Consists of**
voornaam	
achternaam	
geslacht	
geboortedag	
geboortemaand	
geboortejaar	
geboorteplaats	
burgelijke-staat	
e-mail	mail-adres
mobiele-telefoon	
vaste-telefoon	
adres	
woonplaats	
postcode	
website	
hogeronderwijs	instellingsnaam
	afstudeerjaar
middelbare school	schoolnaam
	afstudeerjaar
baan	werkgever
	functie
	huidig
vrienden	vriend-naam

Live Spaces	
Data	**Consists of**
voornaam	
achternaam	
geslacht	
locatie	
geboorte-dag	
geboorte-maand	
geboorte-jaar	
telefoonnummer	
mobiele-nummer	
e-mailadres	
relatie	
geboorteplaats	
opleiding-instelling	
opleiding-examenjaar	
functie	
beroep	
bedrijf	
mailadres-werk	
contacten	contact-naam

Figure A.2: Labels of the data fields that belong to Facebook and Live Space profiles.

Appendix B

Database Schema

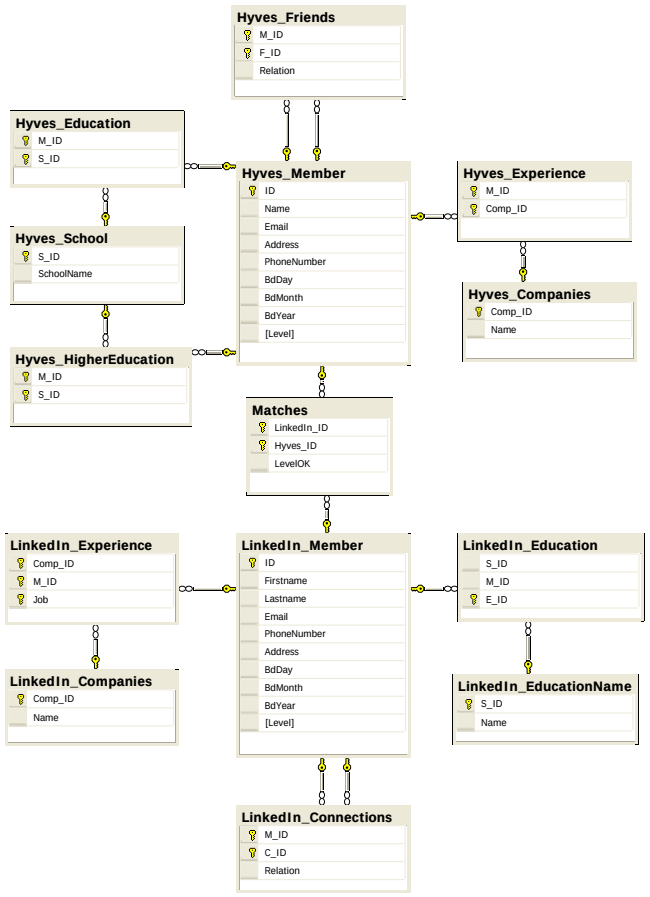


Figure B.1: Database schema after manual schema mapping. This database schema is used by the prototype during the matching process.

Appendix C

Statistics on the Data Sets

In the following tables some numbers on the different attributes of the profiles. We divided the attribute Name into the categories "First name only", "Different name" and "Full name". The category "Different name" includes the cases in which a nickname, an abbreviation, initials or emoticons are used, or in case the name represents more than one person, company, band or email address.

In Table C.1 we listed the statistics for all profiles in the data set. In Table C.2 we listed these numbers for profiles that are part of a match. And in Table C.3 we enlisted the number of attributes completed by both profiles in a true match.

For all tables, total numbers (T) are split over level-1 profiles (L1) and level-2 profiles (L2).

Table C.1: Some numbers on the first data set. Total numbers (T) are split over level-1 profiles (L1) and level-2 profiles (L2).

Attribute	Hyves Profiles						LinkedIn Profiles					
	L1		L2		T		L1		L2		T	
Firstname only	1	(2,1%)	354	(16,9%)	355	(16,3%)	0	(0%)	0	(0%)	0	(0%)
Different name	1	(2,1%)	76	(3,6%)	77	(3,5%)	0	(0%)	17	(0,8%)	17	(0,8%)
Full name	45	(95,7%)	1703	(79,8%)	1748	(80,2%)	91	(100%)	2050	(99,2%)	2141	(99,2%)
Birthday (d-m)	42	(89,4%)	1571	(73,7%)	1613	(74,0%)	11	(12,1%)	100	(4,8%)	111	(5,1%)
Birthday (y)	42	(89,4%)	1571	(73,7%)	1613	(74,0%)	11	(12,1%)	0	(0%)	11	(0,5%)
Email	45	(95,7%)	0	(0%)	45	(2,1%)	87	(95,6%)	233	(11,3%)	320	(14,9%)
Phone	10	(21,3%)	26	(1,2%)	36	(1,7%)	0	(0%)	0	(0%)	0	(0%)
Address	40	(85,1%)	1588	(74,4%)	1628	(74,7%)	91	(100%)	2067	(100%)	2158	(100%)

Table C.2: Some numbers on the matches of the first data set. Total numbers (T) are split over level-1 profiles (L1) and level-2 profiles (L2).

Attribute	**Hyves Profiles**						**LinkedIn Profiles**					
	L1		**L2**		**T**		**L1**		**L2**		**T**	
Firstname only	0	(0%)	0	(0%)	0	(0%)	0	(0%)	0	(0%)	0	(0%)
Different name	0	(0%)	0	(0%)	0	(0%)	0	(0%)	0	(0%)	0	(0%)
Full name	17	(100%)	94	(100%)	111	(100%)	17	(100%)	94	(100%)	111	(100%)
Birthday (d-m)	16	(94,1%)	77	(81,9%)	93	(23,8%)	4	(23,5%)	8	(8,5%)	12	(10,8%)
Birthday (y)	16	(94,1%)	77	(81,9%)	93	(23,8%)	4	(23,5%)	8	(8,5%)	12	(10,8%)
Email	16	(94,1%)	12	(12,8%)	28	(25,2%)	16	(94,1%)	13	(13,8%)	29	(26,1%)
Phone	5	(29,4%)	2	(2,1%)	7	(6,3%)	0	(0%)	0	(0%)	0	(0%)
Address	15	(88,2%)	80	(85,1%)	95	(85,6%)	17	(100%)	94	(100%)	111	(100%)

Table C.3: Some numbers on the matches of the first data set. Total numbers (T) are split over level-1 profiles (L1) and level-2 profiles (L2).

Attribute	Matches					
	L1		L2		T	
Firstname only	0	(0%)	0	(0%)	0	(0%)
Different name	0	(0%)	0	(0%)	0	(0%)
Full name	17	(100%)	94	(100%)	111	(100%)
Birthday (d-m)	4	(23,5%)	6	(6,4%)	10	(9,0%)
Birthday (y)	4	(23,5%)	6	(6,4%)	10	(9,0%)
Email	15	(88,2%)	4	(4,3%)	29	(26,1%)
Phone	0	(0%)	0	(0%)	0	(0%)
Address	15	(88,2%)	80	(85,1%)	95	(85,6%)

Table C.4: The number of schools is completed at Hyves profiles.

Nr of Schools	Hyves		
	L1	L2	T
0	14	668	682
1	11	576	587
2	12	563	575
3	9	220	229
4	1	58	59
5-6	0	30	30
7+	0	18	18

Table C.5: The number of schools is completed at LinkedIn profiles.

Nr of Schools	LinkedIn		
	L1	L2	T
0	1	122	123
1	37	950	987
2	45	664	709
3	6	222	228
4	1	77	78
5	0	16	16
6	1	9	10
7+	0	7	7

Table C.6: The number of Companies is completed at Hyves profiles.

Nr of Companies	Hyves		
	L1	L2	T
0	29	1296	1325
1	12	603	615
2	5	136	141
3	0	55	55
4	0	22	22
5-6	1	11	12
7+	0	10	10

Table C.7: The number of Companies is filled in at LinkedIn profiles.

Nr of Companies	LinkedIn		
	L1	L2	T
0	3	64	67
1	14	306	320
2	18	359	377
3	18	432	450
4	20	317	337
5	10	233	243
6-7	6	373	388
8-9	1	84	85
10+	1	54	55

Table C.8: The number of connections Hyves profiles have.

Nr of Connections	Hyves		
	L1	L2	T
1	10	1983	1993
2	7	105	112
3-6	9	32	41
7+	21	0	21

Table C.9: The number of connections LinkedIn profiles have.

Nr of Connections	LinkedIn		
	L1	L2	T
1	5	1758	1763
2	3	196	199
3-6	12	111	123
7+	71	2	73

Appendix D

More Results from the Experiments

D.1 Distribution of Similarity Scores for True Matches

The graphs below show the distribution of the similarity scores of the true matches, when compared with different string matchers. Only the most interesting string matchers are shown in the graphs.

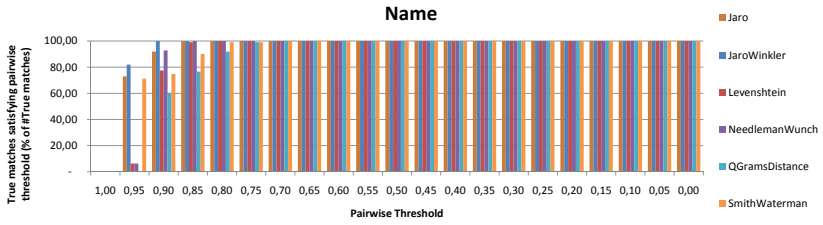


Figure D.1

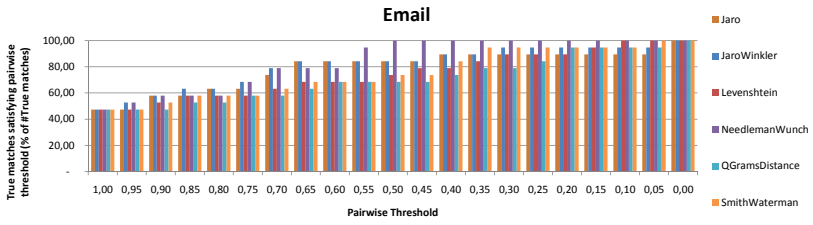


Figure D.2

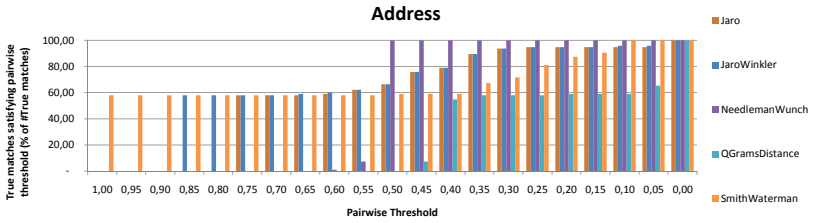


Figure D.3

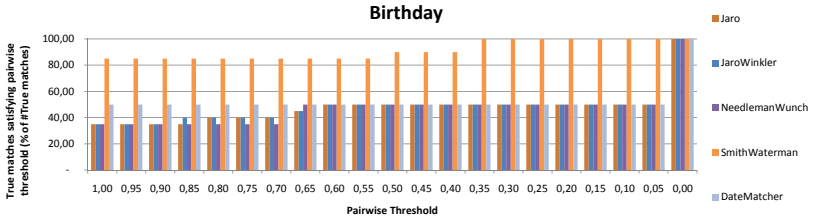


Figure D.4

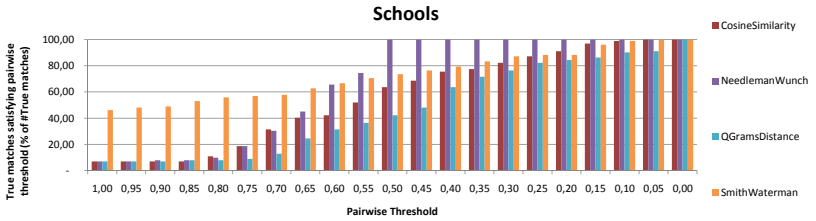


Figure D.5

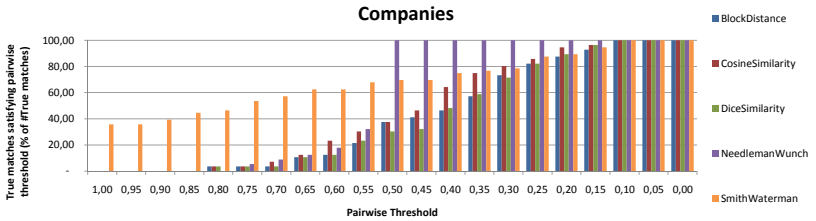


Figure D.6

D.2 Precision and Recall for Different String Matchers

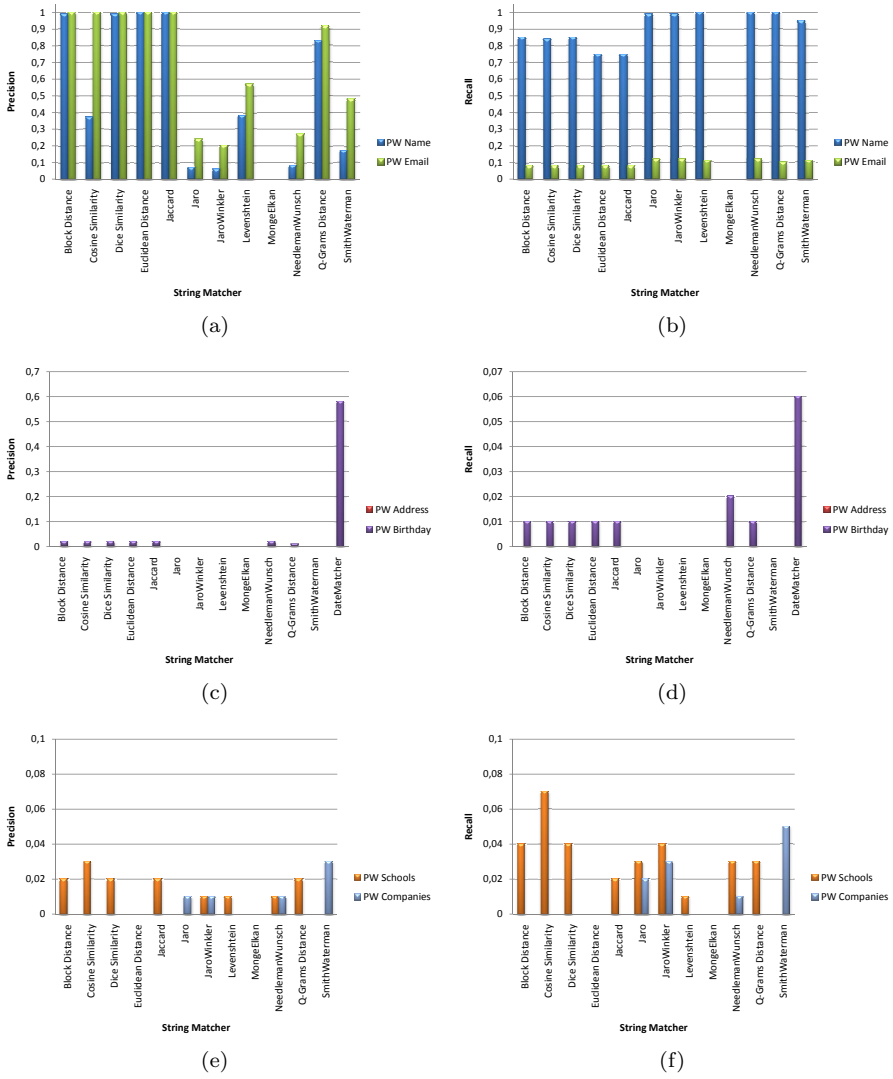


Figure D.7: Precision and Recall for different attributes and for different string matchers, pairwise comparison only *(Method: Normal,* $d_{pw} = 0.70$ *and* $d_{nw} = 0.05$, $\omega_{pw} = \omega_{nw}$*).*

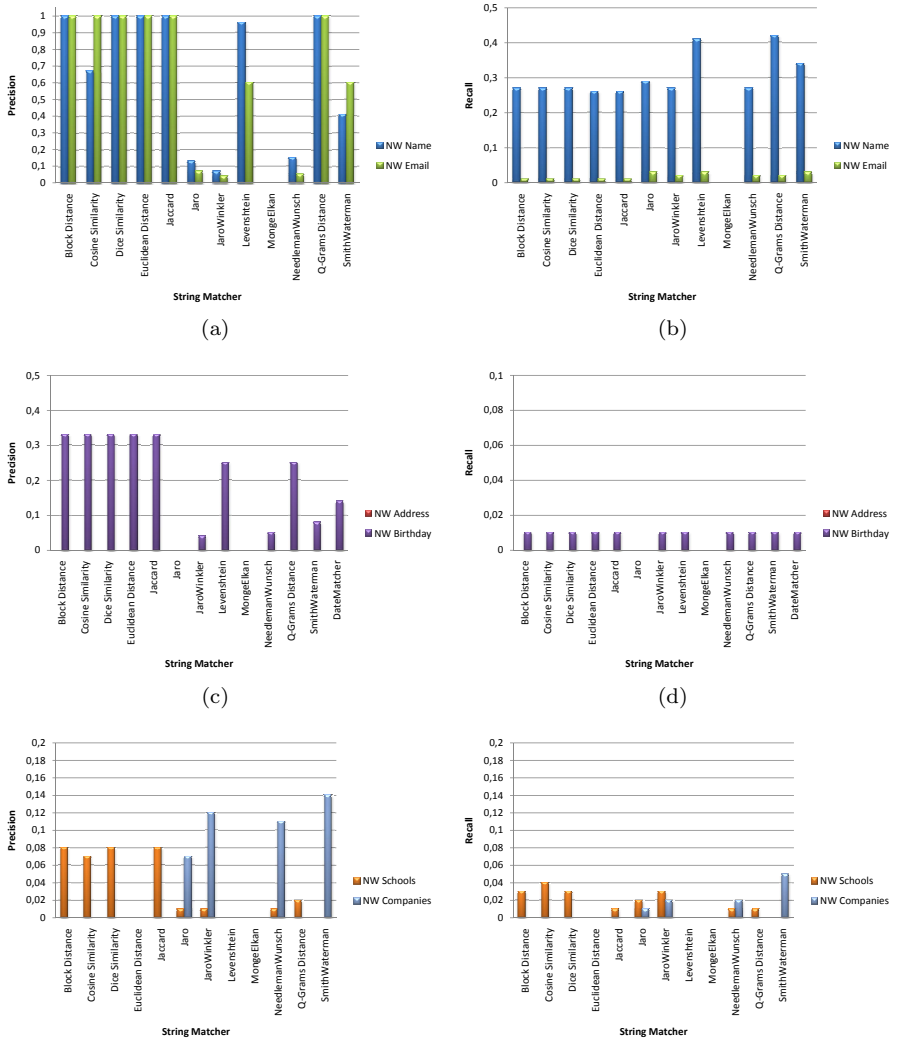


Figure D.8: Precision and Recall for different attributes and for different string matchers, with network comparison *(Method: Normal,* $d_{pw} = 0.70$ *and* $d_{nw} = 0.05$, $\omega_{pw} = \omega_{nw}$*).*

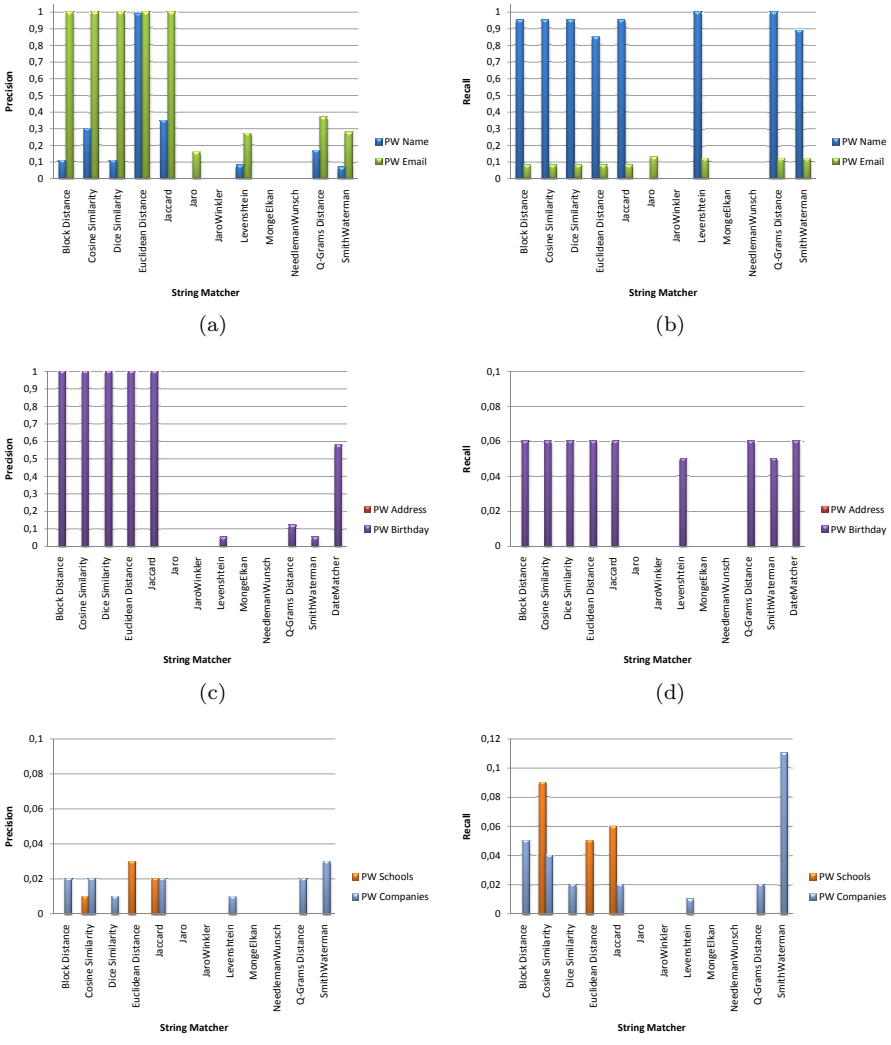


Figure D.9: Precision and Recall for different attributes and for different string matchers, pairwise comparison only *(Method: Normal,* $d_{pw} = 0.50$ *and* $d_{nw} = 0.05$*,* $\omega_{pw} = \omega_{nw}$*).*

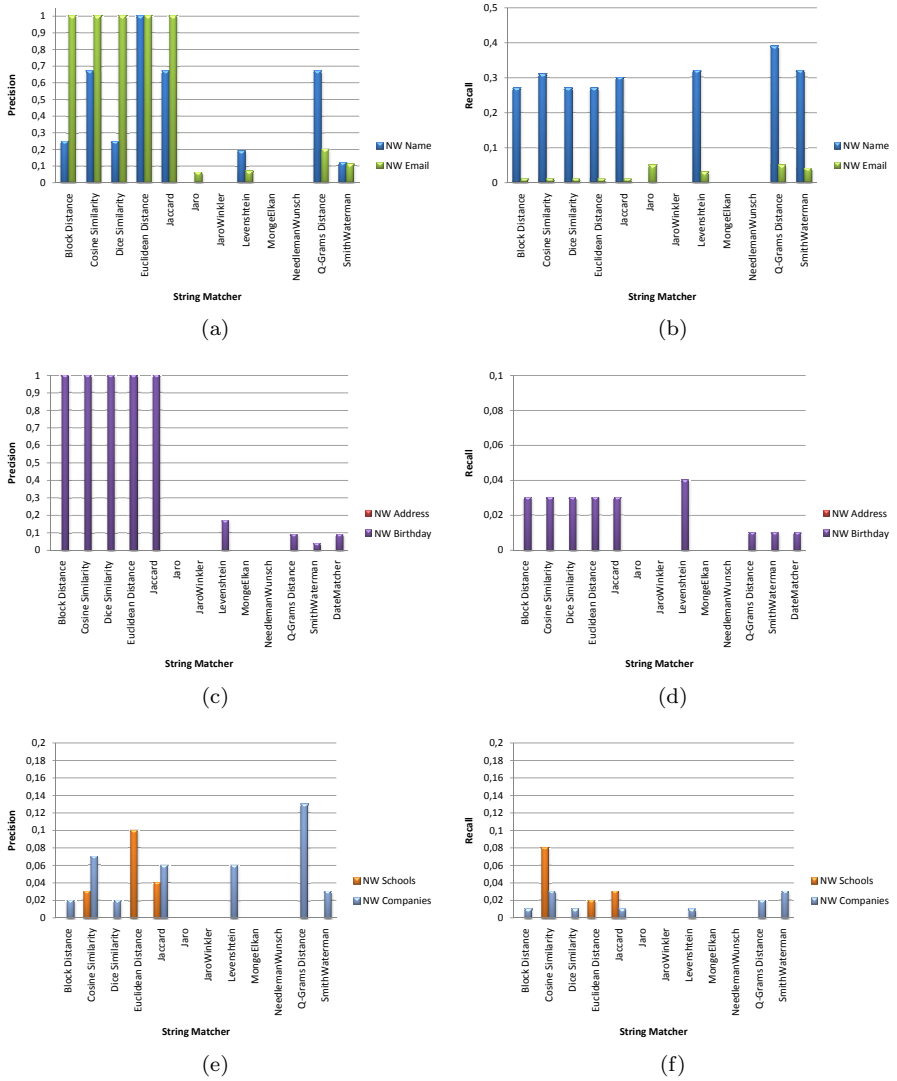


Figure D.10: Precision and Recall for different attributes and for different string matchers, with network comparison *(Method: Normal,* $d_{pw} = 0.50$ *and* $d_{nw} = 0.05$, $\omega_{pw} = \omega_{nw}$*).*

Appendix E

Approximate String Matching Methods

We briefly discussed some approximate string matching methods in Section 3.1.1. In this section we will discuss these methods in more detail.

E.1 Levenshtein Distance

The Levenshtein distance, or edit-distance, measures the minimal number of transpositions, inserts and deletions of characters are needed to transform one string in the other.

As an example we present the way the edit-distance is calculated for the strings s_1 'entity' of length $n = |s_1| = 6$ and string s_2 'identity' of length $m = |s_2| = 7$. We create a matrix d of dimension $(n+1) \times (m+1)$. The first row and column are filled with the numbers [0..6] respectively [0..7], in general:

$$\begin{aligned} d[0][i] &= i, \quad 0 < i \leq m \\ d[j][0] &= j, \quad 0 < j \leq m. \end{aligned} \tag{E.1}$$

Now every other entry $d[i][j]$ can be filled with respect to the previous row and column.

$$\begin{aligned} cost &= 0, \quad d[i][0] = d[0][j] \\ cost &= 1, \quad otherwise. \end{aligned} \tag{E.2}$$

Each entry $d[i][j]$ is then

$$\begin{aligned} d[i][j] = min(&d[i-1][j] && +1, \\ &d[i][j-1] && +1, \\ &d[i-1][j-1] && +cost). \end{aligned} \tag{E.3}$$

The result of the method is the value of $d[n][m]$. In case of the example, this is 2.

An edit-distance of 0 means that no insertions, deletions or substitutions are needed, hence the words are equal. Every number higher means one more action is needed to transform one word in the other. The maximal distance is the $max(n, m)$.

Table E.1: distance matrix

		e	**n**	**t**	**i**	**t**	**y**
	0	1	2	3	4	5	6
i	1	1	2	3	3	4	5
d	2	2	2	3	4	4	5
e	3	2	3	3	4	4	5
n	4	3	2	3	4	5	6
t	5	4	3	2	3	4	5
i	6	5	4	3	2	3	4
t	7	6	5	4	3	2	3
y	8	7	6	5	4	3	2

The Levenshtein distance is suitable for the detection of misspellings, but when it is applied on a complete sentence instead of the tokens separately, it will perform very badly in case of different word order and abbreviations.

E.2 Jaro Distance

The Jaro distance d_j is expressed in the number of matching characters m and the number of transpositions t. A transposition is the swapping of two letters. Two matching characters are considered matching only if they are no farther than

$$\left\lfloor \frac{max(|s_1|, |s_2|)}{2} \right\rfloor - 1, \tag{E.4}$$

e.g. they must be within a half word distance of each other. If we know the number of matches and the number of transpositions, we can calculated the Jaro distance d_j

$$d_j = \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m} \right). \tag{E.5}$$

For the example words s_1 'entity' and s_2 'identity' with $m = 6$, $|s_1| = 6$, $|s_2| = 8$ and $t = 0$ the Jaro distance would be

$$d_j = \frac{1}{3}\left(\frac{6}{6} + \frac{6}{8} + \frac{6-0}{6} \right) \approx 0.917. \tag{E.6}$$

The range of the Jaro distance is $d_j \in \langle 0, 1]$. If it evaluates to 1, it is a complete match, and the lower the value, the less similarity between the words.

Jaro is suitable for words in which misspellings consist of swapping of characters. Where a word with swapped characters would need two substitutions in the Levenshtein distance, here it would count as one transposition (if they are within half the length of the longest word).

E.3 Jaro-Winkler Distance

The Jaro-Winkler distance is a variant of the Jaro distance which assigns better ratings to two words s_1 and s_2 if they have a shared prefix of length l. The length l is maybe maximal 4 characters. p is a scaling factor that assigns a

weight for accounting the prefix in the metric. The standard value is $p = 0.1$. The Jaro-Winkler distance d_w is calculated as follows:

$$d_w = d_j + lp(1 - d_j). \tag{E.7}$$

The example words s_1 and s_2 do not share a prefix, so in this case $d_w = d_j$. But if we take the words s_1 'Johnson' and s_2 'Jonhsons' the Jaro distance would be

$$d_j = \frac{1}{3}\left(\frac{6}{6} + \frac{6}{8} + \frac{6-1}{6}\right) \approx 0.861 \tag{E.8}$$

and the Jaro-Winkler distance

$$d_w \approx 0.861 + 2 \times 0.1(1 - 0.861) \approx 0.888. \tag{E.9}$$

Notice that the value of d_w is slightly higher then d_j, because the first two characters match.

The Jaro-Winkler distance has the same range of values as the Jaro-distance. However, it is better suited for words that share a prefix. With only minor changes to the algorithm, this could also work for suffixes. Giving higher scores to prefixes could help match abbreviations, such as 'Inc.' and 'Incorporated'.

E.4 Jaccard

The Jaccard measure simply measures the ratio of equal tokens in the union of tokens of both strings. Thus, we have a set of tokens for string s_1, $S_1 = Tokenize(s_1)$ and the same for s_2, $S_2 = Tokenize(s_2)$, then

$$Jaccard(s_1, s_2) = \frac{|S_1 \bigcap S_2|}{|S_1 \bigcup S_2|}. \tag{E.10}$$

The range of Jaccard is $Jaccard(s_1, s_2) \in \langle 0, 1]$ as well. With 1 being an exact match. Unfortunately, misspellings are counted as different words and hence can decrease the score enormously. Moreover, it matches all kinds of words, even words that occur often, such as 'the' and 'a'. The following method compensates for this.

E.5 Cosine Similarity and TF/IDF

Cosine similarity comes from the information retrieval field and is used for determining to which extent two documents A and B are similar, based on a set of distinct terms T. A document in this case could be a sentence containing several words. Terms do not exactly mean words, better would be tokens, since they split words on each character that is not alpha-numeric. For instance, 'j.h.smith@provider.com' would result in the set of tokens $j, h, smith, provider, com$. If we have $k = |T|$ terms, then the space we are talking about is k-dimensional. Each document is a vector in this space expressing for each term how many times the document consists this term. Then, the cosine similarity is

$$Sim(A, B) = cosine\theta = \frac{A \bullet B}{|A||B|}. \tag{E.11}$$

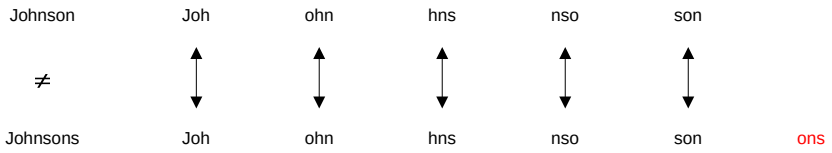


Figure E.1: Example of Q-Grams

A problem with this is that some terms are not that discriminative for a certain document and hence disturb the measure. A solution is provided with TF/IDF.

TF/IDF stands for Term Frequency/Inverse Document Frequency. It compensates for words like 'a', 'have' and 'is', or in the case of mail addresses 'com', that are very common in each document.

The term frequency $tf_{i,j}$ for a term i in a document d_j is

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}, \tag{E.12}$$

with $n_{i,j}$ being the number of times that term i occurs in document $d_{i,j}$.

The inverse document frequency idf_i measures how important a term i is for the set of documents D:

$$idf_i = log\frac{|D|}{|\{d : t_i \in d\}|}, \tag{E.13}$$

with $d \in D$.

The TF/IDF value is then

$$tfidf_{i,j} = tf_{i,j} \times idf_i. \tag{E.14}$$

This method is very suitable for detecting differences in word order. For instance, in data fields containing the full name of a person. Consider a field 'John Smith' in one source and 'Smith, John' in another. This method would conclude they are the same. Unfortunately, this method cannot detect misspellings. 'John' and 'Jonn' would be seen as two complete different words. This is undesired.

E.6 Q-grams

An approach that can deal with the problem mentioned above is Q-Grams. Q-Grams divide each word in every possible sequence of a small number of subsequent characters. See the example in Figure E.1. On these substrings the cosine similarity algorithm can be applied. Here the words 'Johnson' and 'Johnsons' are split in Q-Grams of length $q = 3$. Because 'Johnsons' consists of one character extra, it also will end up with one Q-Gram more. This Q-Gram cannot be matched with another one. [11]

Words that are very similar share many infrequent Q-Grams.